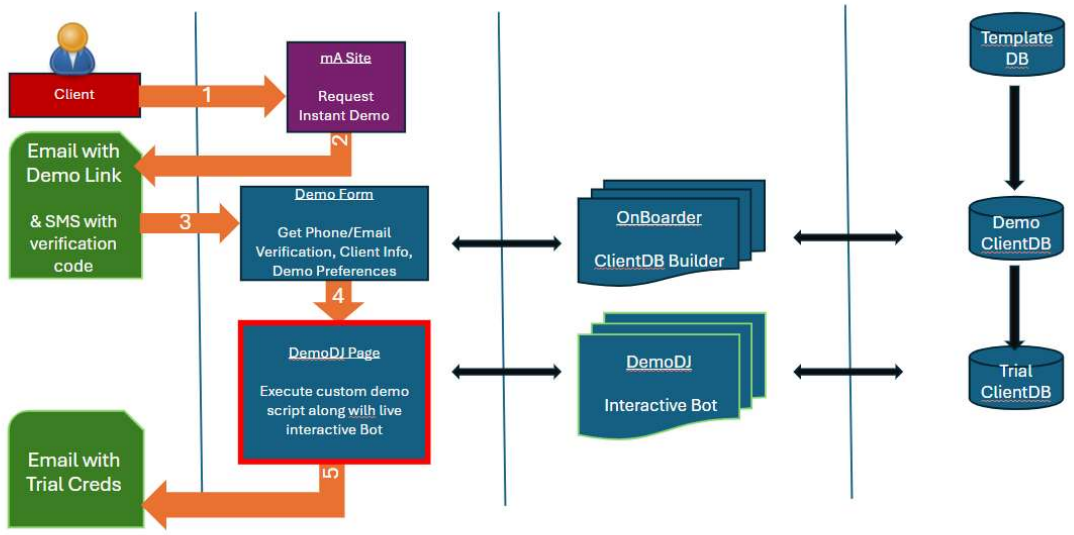
AutoDemo : A feature that lets your app Demo itself!

User Experience & High-level Components



AutoDemo : A feature that lets your app Demo itself!

User Experience & High-level Components

Compponnt	Details	Work Expected
MA Site	MagicAudit website	Provide link in site that accepts client request for quick demo. Enduce site visitors to take this quick instant demo. We get validated email and phone number in return at the least
Demo Link Email	Email Text inviting to demo	Customized email text to prompt client to click demo link
Demo Form	A new web form to accept more details in order to tailor the demo. This may include functionality focus, role etc	Design the new form, detailed but compact. Client shouldn't be intimidated to complete it
DemoDJ Page	The main page where demo is orchestrated by the interactive bot	UI refinement of bot overlay
Trial Link Email	Email text informing about trial access	Details and terms of trial need to be identified
OnBoarder Class/Agent	Python class that takes client details and created a client datbase tailored for the demo	Client customization can be enhanced
DemoDJ Clas/agent	Pytho class that exceuts the dmo scriptas well a iteracts with live user	Development of proper demo script
Trial DB	Wel curated Demo DB template	Build a good demo org db with projects, evals and reports. Dashboard should look wel fleshed
ClientDB	Client customized DB from temlate DB	Identify ways to enhance client context
TrialDB	Inal trial DB for the client	Identify if there are more options needed

Onboarding
Business Requirement Document (BRD)

Project Name	magicAudit
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

The onboarding process is designed to introduce prospective clients to the system through a structured, efficient, and supportive approach that builds client confidence and encourages a seamless transition from a demo to trial to subscription model. This onboarding process will cover all stages from the initial demo to trial access, user account setup, and transition to a fully active account. The aim is to ensure customer satisfaction, reduce friction, and establish a reliable process that can be scaled with business growth.

2. Business Objectives

2.1. Increase Client Conversion Rates: Provide a robust, guided onboarding process to enhance demo and trial conversion into paying clients.
2.2. Improve Customer Satisfaction and Retention: Streamline setup, minimize onboarding errors, and enhance customer experience to foster trust and long-term engagement.
2.3. Standardize Onboarding Process: Create a scalable, standardized onboarding workflow that can adapt to changes in pricing models or subscription types without disrupting client experience.
2.4. Efficient Resource Management: Enable an "outboarding" mechanism to release resources for clients who choose not to proceed after their trial period, optimizing resource utilization.

3. Requirements

3.1. Functional Requirements

3.1.1. Demo Creation and Scheduling

3.1.1.1. The system should allow the creation of customized demos tailored to prospective client needs.

3.1.1.2. The option to schedule and manage demos with prospective clients.

3.1.2. Trial Access Setup

3.1.2.1. Provide seamless transition from demo to trial access with a single-click activation of trial accounts.

3.1.2.2. Enable creation of trial accounts and user profiles for client users.

3.1.3. User Account and Project Setup

3.1.3.1. Allow easy setup of user accounts and assignment of appropriate roles and permissions.

3.1.3.2. Enable clients to create initial projects within the trial period to familiarize them with key features.

3.1.4. Transition to Full Account

3.1.4.1. Support the process to convert trial accounts to fully active, paid accounts.

3.1.4.2. Generate billing information and initiate invoicing upon transition.

3.1.5. Data Cleanup for Outboarding

3.1.5.1. Automate data cleanup for clients who decide not to continue, including removal of user accounts and release of resources.

3.2. Non-Functional Requirements

3.2.1. Reliability

3.2.1.1. Ensure a 99.9% uptime for onboarding-related services to avoid disruptions.

3.2.2. Scalability

3.2.2.1. Design the onboarding process to handle a growing volume of users and projects without degradation in performance.

3.2.3. Security

3.2.3.1. Protect client data with industry-standard encryption and secure access protocols throughout the onboarding and outboarding processes.

3.2.4. Usability

3.2.4.1. Provide an intuitive user interface for onboarding steps, ensuring users of all technical levels can complete the process with minimal support.

3.3. Business Rules

3.3.1. Onboarding Period

3.3.1.1. Clients are allowed a maximum trial period of 7/15 days from activation, after which the trial will expire unless converted to a paid account.

3.3.2. Account Conversion

3.3.2.1. Only clients with valid payment information can transition from a trial account to a paid subscription.

3.3.3. Outboarding Rule

3.3.3.1. If a client chooses not to proceed after the trial, all associated data will be removed within 3 days of trial expiration.

3.3.4. Role-Based Permissions

3.3.4.1. During the onboarding, users will be assigned specific roles and permissions as determined by their trial or subscription type, limiting access to certain functionalities based on role.

4. Assumptions & Constraints

4.1. Assumptions

4.1.1. The prospective client has an initial understanding of the system and is interested in exploring it through a trial.

4.1.2. The system is set up to automate the conversion from trial to paid accounts without extensive manual intervention.

4.1.3. All users have access to the necessary hardware and internet connectivity for a smooth onboarding experience.

4.2. Constraints

4.2.1. Onboarding must be compatible with existing client systems and must not require additional hardware.

4.2.2. Any changes to subscription pricing or features should not disrupt the existing onboarding process.

4.2.3. Limited support resources are available, so the onboarding process must be intuitive and require minimal customer support intervention.

Public Facing Site
Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. Comprehensive platform for users to explore and engage with audit services.
1.2. Access to evaluations and API tools with detailed documentation.
1.3. User-friendly interface with flexible subscription plans: Free, Standard, and Enterprise.
1.4. Evaluation Store for on-demand audit toolsets & services.
1.5. Streamlined registration process with automated API token generation.
1.6. Scalable design to support future growth and expansion.
1.7. Efficient access and management of Magic Audit's offerings.

2. Business Objectives

2.1. Provide an online platform to market Magic Audit's app and services.
2.2. Offer seamless access to API documentation and token generation.
2.3. Facilitate subscription plans for different types of users (Free, Standard, Enterprise).
2.4. Create an Evaluation Store for users to explore different evaluation types.

3. Requirements

3.1. Functional Requirements

3.1.1. Landing Page:

3.1.1.1. The landing page will prominently present an overview of the Magic Audit platform, highlighting its core offerings such as evaluations (SME, interview-based, and algorithmic), subscription plans, and API services.

3.1.1.2. Users will have quick access to important sections like **Pricing**, **API Documentation**, and the **Evaluation Store**, all of which will be available via the site's header and footer.

3.1.2. User Registration:

3.1.2.1. Allow users to register on the site.

3.1.2.2. Provide API tokens post-registration for accessing platform features.

3.1.2.3. Subscription type selection during the registration process.

3.1.3. Subscription Plans:

3.1.3.1. Three Subscription Tiers:

Offer three distinct subscription plans to users:

3.1.3.1.1. Free Plan: Limited access to evaluation services with restrictions on certain functionalities. It can also provide all the access but that will only be a free trial of 7 days.

3.1.3.1.2. Standard Plan: Broader access to platform modules, with some exclusions such as the Studio.

3.1.3.1.3. Enterprise Plan: Full access to the entire application, including the Magic Audit Studio, API services, and all evaluation types.

3.1.3.2. Free Plan Features:

The Free Plan provides basic access to evaluations and limited API usage, targeting individuals or businesses interested in exploring the platform before committing to higher plans.

3.1.3.3. Standard Plan Features:

This plan grants access to most modules, except the advanced features like Studio, and includes higher limits for evaluations and API requests.

3.1.3.4 Enterprise Plan Features:

The Enterprise Plan offers unrestricted access to all of Magic Audit's functionalities, including full API and evaluation service usage without limitations.

3.1.3.5. Subscription Validation During Registration:

The system will verify the selected subscription type during the registration process and configure user access, accordingly, ensuring users receive the correct level of service based on their selection.

3.1.4. Evaluation Store:

3.1.4.1. A specific section on the site will feature the **Evaluation Store**, where users can explore different types of evaluations based on their needs.

3.1.4.2. Evaluations will be organized by topic (e.g., **Interview**, **SME**, and **Score**, allowing users to filter and select services & tools that match their requirements.

3.1.4.3. The store will provide the ability to showcase features of various audit tool sets available and allow existing users to download these tools, like apps, within their existing MagicAudit instance.

3.1.4.4. The store is designed with scalability in mind, with plans to potentially evolve it into a standalone application that offers more comprehensive evaluation services in the future.

3.1.5. API Documentation:

3.1.5.1. Provide detailed API reference and documentation for developers.

3.1.5.2. Users can either use the full app or specific APIs based on their requirements.

3.2. Non-Functional Requirements

3.2.1. Security: Ensure secure handling of user data and API token generation.

3.2.2. Performance: The website should handle high traffic and multiple concurrent requests seamlessly.

3.2.3. Scalability: The platform should be scalable to handle future enhancements, such as additional evaluation types or expansion of the Evaluation Store.

3.2.4. User Experience: The site should be intuitive, with a clean layout and easy navigation across pages.

3.3. Business Rules

3.3.1. Users must select a subscription plan during registration.

3.3.2. Only users with valid API tokens can access the API functionalities.

3.3.3. Evaluation purchases are restricted to registered users.

4. Assumptions & Constraints

4.1. The API token generation will be automated post-registration.

4.2. The Evaluation Store will initially be part of the PFS but may evolve into a separate entity.

4.3. The site will require regular updates as new evaluation types or subscription plans are added.

Authentication and Administration

Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Sharon Pala
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The **Authentication and Administration** module of the MagicAUDIT application is critical for safeguarding the system's security, ensuring efficient user management, and providing seamless access to authorized users.

1.2. This module is designed to protect sensitive user information, enhance operational control, and improve user experience by implementing secure, role-based access.

1.3. The system will ensure clear segregation of duties, strict access controls, and enhanced data integrity measures through the use of tokens and session management.

2. Business Objectives

The Authentication and Administration module aims to enhance security, streamline user management, and improve usability, all while enforcing role-based permissions.

2.1. Enhance Security

2.1.1. Secure user access and data through token-based authentication and session tracking.

2.1.2. Encrypt user data at rest and in transit to protect privacy.

2.2. Streamlined User Management

2.2.1. Implement role-based access control (RBAC) to restrict access by role.

2.2.2. Allow administrators to modify roles and permissions dynamically.

2.3. Improve Usability

2.3.1. Ensure a smooth login/logout process.

2.3.2. Enhance session management for better user experience.

2.3.3. Provide simplified access to features based on roles.

2.3.4. Design an intuitive UI to minimize the learning curve.

2.4. Single Company Installation

2.4.1. Support only one company per installation to simplify data handling.

2.4.2. Reduce cross-company data leakage risk and ease administration.

3. Requirements

3.1. Functional Requirements

3.1.1. User Authentication

3.1.1.1. Registration/Login Process:

3.1.1.1.1. Users will provide four required fields: username, password, contact name, and client or company name during login.

3.1.1.1.2. Input Validation:

3.1.1.1.2.1. The system will validate these fields against predefined security standards.

3.1.1.1.2.2. Passwords must meet complexity requirements, including:

3.1.1.1.2.2.1. Minimum length

3.1.1.1.2.2.2. Use of special characters

3.1.1.1.2.2.3. Periodic expiration (Future enhancement)

3.1.1.1.3. Upon successful login, a JSON Web Token (JWT) or OAuth token will be generated to identify and authenticate the user during subsequent interactions.

3.1.1.2. Multi-Factor Authentication (MFA): (Future Enhancement)

3.1.1.2.1 As an additional security layer, MFA will be enforced for high-privilege users.

3.1.1.2.2 MFA will require users to verify their identity using a secondary authentication factor such as:

3.1.1.2.2.1 A one-time password (OTP)

3.1.1.2.2.2 An authenticator app

3.1.1.3. Token Utilization:

3.1.1.3.1. The generated token will be used in the headers of all API requests to validate each call and prevent unauthorized access.

3.1.1.3.2. Session Expiry:

3.1.1.3.2.1. Tokens will have a predefined expiry time.

3.1.1.3.2.2. The system will prompt users to re-authenticate once the token is invalidated. (?)

3.1.1.3.2.3. A token refresh mechanism will be included for seamless re-authentication.

3.1.1.4. Logout Behavior:

3.1.1.4.1. Users will have the capability to log out of the application, which will invalidate their current session and token to prevent unauthorized access.

3.1.1.4.2. A confirmation message will be provided to the user upon successful logout, ensuring clarity regarding their session status.

3.1.2. Administration

3.1.2.1. Overview of the Administration Workflow:

The administration module is crucial in ensuring that the appropriate user roles and access controls are in place to maintain the system's integrity and efficiency. The workflow is designed to manage the assignment of user roles, modify permissions, enforce single company restrictions, and monitor session management. By having clear administrative structures, this module will allow secure, flexible, and streamlined management.

3.1.2.2. User Roles:

The system will have three distinct user roles, each with specific access privileges based on their responsibilities:

3.1.2.2.1. Bench (Governance Model User):

The Bench role, or Governance Model User, plays a crucial role in overseeing the governance and compliance processes within the system. They ensure AI models and projects comply with internal policies, legal standards, and ethical guidelines. With special access to governance features, these users can review, monitor, and audit data throughout the AI project lifecycle. Bench users can also add new projects and models, facilitating streamlined governance. They can engage in three levels of testing:

3.1.2.2.1.1 Objective Testing: This involves assessing input files to generate score metrics based on performance criteria. Users can evaluate how well the AI models function according to predefined standards.

3.1.2.2.1.2 Reflective Testing: Here, users can create their interview scenarios and receive scores from the AI, providing insights into the effectiveness of the models in real-world applications.

3.1.2.2.1.3 Investigative Testing: This allows for assignments to Subject Matter Experts (SMEs) who can offer feedback on the project, further enhancing its reliability and compliance.

Bench users ensure governance while actively improving AI models by tracking regulatory changes and maintaining alignment with ethical and legal standards

3.1.2.2.2 SME Module Users:

SME Module Users have a dynamic role that extends beyond managing domain-specific operations. Their responsibilities:

3.1.2.2.2.1 Scoring Assigned Questions: SMEs evaluate specific projects by assigning scores (on a scale from 1 to 10) to pre-set questions designed to assess different aspects of the project.

3.1.2.2.2.2 Providing Feedback: In addition to scoring, SMEs offer feedback on the project, providing insights into areas for improvement, highlighting strengths, and suggesting necessary adjustments.

3.1.2.2.2.3 Adding Custom Questions: If needed, SMEs can create additional questions tailored to the project's unique requirements and provide corresponding scores/feedback to enhance the evaluation process.

3.1.2.2.2.4 Self-Evaluation: SMEs can conduct self-evaluations on their assigned projects, applying the same process of scoring, providing feedback, and adding custom questions. This self-assessment adds another layer of comprehensive evaluation, allowing SMEs to reflect on their own contributions and project progress.

3.1.2.2.2.5 AI-Enhanced Brainstorming: The "Improve" button provides SMEs with a range of interactive tools to refine project assessments:

3.1.2.2.2.5.1 Chat: An interactive messaging feature that allows SMEs to communicate with the AI in real-time. This helps in clarifying ideas, sharing insights, and collaboratively refining project assessments directly within the brainstorming session.

3.1.2.2.2.5.1 Call: Facilitates audio conversations with the "RAG model" for real-time, in-depth discussions and model-based insights.

3.1.2.2.2.5.2 Whiteboard: A space for visual interaction that allows SMEs to illustrate ideas and collaboratively enhance project evaluations.

3.1.2.2.2.5.3 Auto Optimize: This prompt refinement tool enables SMEs to specify objectives and outline prompt requirements, letting AI iteratively refine the prompt to generate the most effective version. SMEs can then incorporate this optimized prompt into the broader feedback loop for comprehensive project evaluation.

This combination of tools supports a more collaborative, AI-boosted approach to project insights and enhancements.

3.1.2.2.3. Dev Users:

Developers have a critical role in managing and improving the overall application. They have full access to modify settings, handle performance configurations, and implement system updates. Their primary responsibilities include:

3.1.2.2.3.1 Maintaining and Enhancing Features: Developers work on writing, testing, and deploying new functionalities, such as updates to performance settings and managing the behavior of the score calculation. If any setting is changed, they must ensure that the scoring algorithm behaves correctly at each level.

3.1.2.2.3.2 Security and Access: They ensure that all users who access the system are valid and registered, as unregistered users cannot log in. They also implement API calls using secure tokens, which are necessary for fetching client data and ensuring only valid, registered users can interact with both the UI and APIs.

3.1.2.2.3.3 Configuring Settings: Developers have the ability to adjust system settings. For instance, they must maintain the "No Bad Words" file, which is used to filter out inappropriate language. They can update or change this file as needed to ensure compliance.

3.1.2.2.3.4 Collaborating with Other Modules: Developers need to integrate their work with other modules, such as the Bench, SME, and Score APIs, ensuring seamless collaboration and communication across these areas.

Their role is essential for ensuring the application runs smoothly, with a focus on scalability, security, and the efficient handling of configurations specific to each project they are involved in.

3.1.2.3. Access Controls:

Access controls will be strictly enforced based on the defined roles. The system will ensure that users can only access functions and data that align with their specific roles, preventing unauthorized access and maintaining security.

3.1.2.3.1 Role-Based Access:

The system will rely on Role-Based Access Control (RBAC) to ensure that permissions are tailored to the user's role. Each user role will have pre-defined capabilities, which administrators can update as needed.

3.1.2.3.2 Administrator Permissions:

Administrators will have the ability to modify roles and permissions dynamically. This will allow the system to remain flexible while still ensuring strict control over user access and system functions. Administrators can add or revoke permissions depending on evolving business needs, enhancing operational flexibility.

3.1.2.4. Single Company Restriction:

To maintain focus and simplify management tasks, the system will enforce a policy allowing only one company per installation. This means that for each system instance, only a single company can be managed, preventing data mixing between multiple organizations.

3.1.2.4.1. Data Isolation: All system settings, data, and reports will be isolated for the single company being managed, ensuring there is no risk of data leakage or contamination between different entities.

3.1.2.4.2. Focused Administration: The single company restriction simplifies administrative tasks by keeping the focus solely on the operations of one company, making data handling and reporting more straightforward.

3.1.2.5. Reporting and Data Retrieval:

The administration module will play a key role in data reporting and retrieval. Upon successful login, the system will fetch, and present relevant company data based on the logged-in user's role. This will ensure that users are provided with the information they need, without access to unauthorized data.

3.1.2.6. Company Information Retrieval:

Following user authentication, the system will automatically retrieve key company information, such as the company name, reports relevant to the user, and other data based on their role. This retrieval process ensures that users have immediate access to the data necessary for their role.

3.1.2.7. Role-Based Data Presentation:

The homepage will present data tailored to each user's role to enhance operational efficiency and ensure relevant information is readily accessible. For example:

3.1.2.7.1 Bench Users (Governance Model Users): They will receive governance-related reports and insights necessary for monitoring compliance, overseeing data management, and ensuring adherence to industry regulations. This enables them to conduct risk assessments and maintain security protocols effectively.

3.1.2.7.2 SME Module Users: These users will have access to project-specific data that allows them to evaluate performance, provide feedback, and conduct self-assessments. They can score assigned questions and create custom queries to address unique project requirements, ensuring comprehensive evaluations.

3.1.2.7.3 Development Users: Developers will be provided with data regarding application performance, system configurations, and feature updates. This information will support their efforts in writing, testing, and deploying new functionalities while maintaining application integrity.

3.1.2.8. Session Management:

The system will include robust session management functionality. Every user session will be tracked in real time, capturing essential details like login time, last activity, and total session duration.

3.1.2.8.1. Session Tracking: The system will record all active sessions and monitor activity for security purposes. This tracking will allow administrators to audit user activity and ensure compliance with security policies.

3.1.2.8.2. Session Termination by Administrators: Administrators will have the authority to terminate user sessions if suspicious activity is detected. This will enhance security by allowing prompt action to be taken in case of unauthorized activity.

3.2. Non-Functional Requirements

3.2.1. Performance:

The administration module will be optimized to support a scalable number of concurrent users, ensuring that there are no performance issues during high-demand periods.

3.2.1.1. Concurrency Support: The system must be able to handle a minimum of X concurrent administrative users without degrading performance. This ensures that administrative tasks can be carried out efficiently, even in a large-scale deployment.

3.2.1.2. Response Time: Administrative actions, such as assigning roles, changing permissions, or generating reports, must have a response time that does not exceed X seconds. This will ensure that administrative processes are not delayed due to system lags.

3.2.2. Usability:
The administration module will be designed with an intuitive and user-friendly interface to ensure that administrators and other privileged users can manage roles and permissions with ease.

3.2.3. User-Friendly Interface: The interface will be designed to minimize the learning curve for administrators. They should be able to easily navigate through user management options, adjust permissions, and retrieve reports without requiring extensive training.

3.2.4. Security:
Security measures will be enforced throughout the administrative module, ensuring that sensitive data is protected, and user roles are accurately enforced.

3.2.4.1. Access Control Security: Only users with administrative privileges will be able to alter roles or permissions. The system will log all changes to user roles and access controls for auditing purposes.

3.2.4.2. Data Confidentiality: Data in the administration module, especially concerning user roles, permissions, and company data, will be protected using encryption. This will ensure that even if unauthorized access is attempted, the data remains secure.

3.3. Business Rules for Authentication and Administration

3.3.1. User Authentication

3.3.1.1. User Registration/Login:

3.3.1.1.1. Users must provide a valid username, password, contact name, and client or company name during registration and login.

3.3.1.1.2. Passwords must meet complex requirements, including a minimum length, use of special characters, and periodic expiration.

3.3.1.1.3. Tokens (JWT/OAuth) will be generated upon successful login and will be required for all subsequent user interactions.

3.3.1.2. Multi-Factor Authentication (MFA):

3.3.1.2.1. MFA is mandatory for high-privilege users and must involve a secondary authentication factor, such as OTP or authenticator apps.

3.3.1.2.2. MFA verification is required at each login session for high-privilege users.

3.3.1.3. Token Expiration & Re-authentication:

3.3.1.3.1. Tokens will have a fixed expiration time and will require users to re-authenticate once they expired.

3.3.1.3.2. The system will prompt users to refresh tokens or log in again when the session expires.

3.3.1.4. Logout Behavior:

3.3.1.4.1. Logout actions will immediately invalidate the user's token and terminate the session.

3.3.1.4.2. Users will receive a confirmation message upon successful logout.

3.3.2. Administration

3.3.2.1. Role-Based Access Control (RBAC):

3.3.2.1.1. Users can only access features and data allowed by their assigned roles.

3.3.2.1.2. Role changes must be initiated and approved by system administrators.

3.3.2.2. User Roles:

3.3.2.2.1. Bench (Governance Model Users) have access to governance-related features, audits, and compliance workflows.

3.3.2.2.2. SME Users can access features relevant to their domain expertise and approve workflow changes within their module.

3.3.2.2.3. Dev Users (Studio) have full access to development and maintenance features.

3.3.2.3. Dynamic Role & Permission Adjustments:

3.3.2.3.1. Administrators can add, remove, or adjust user roles and permissions dynamically.

3.3.2.3.2. Changes to roles and permissions must be logged for audit purposes.

3.3.3. Single Company Installation

3.3.3.1. Single Company Support:

3.3.3.1.1. The system will enforce the rule of one company per system installation, preventing data from multiple companies being managed together.

3.3.3.1.2. Data isolation will be strictly enforced to prevent cross-company data leakage.

3.3.3.2. Focused Administrative Tasks:

3.3.3.2.1. The administration panel will be optimized for managing a single company, simplifying data reporting and role management.

3.3.4. Session Management

3.3.4.1. Session Tracking:

3.3.4.1.1. All active user sessions will be monitored in real-time for security auditing.

3.3.4.1.2. Session activity logs will include login time, session duration, and last activity timestamp.

3.3.4.2. Administrator Session Controls:

3.3.4.2.1. Administrators will have the ability to terminate any active session in case of suspicious activity.

3.3.4.2.2. Terminated sessions will be logged for security purposes.

3.3.5. Reporting and Data Retrieval

3.3.5.1. Role-Based Data Access:

3.3.5.1.1. Upon login, users will only have access to reports and data relevant to their specific roles.

3.3.5.1.2. Role-based dashboards will present personalized data to users, improving operational efficiency.

3.3.5.2. Company Information Display:

3.3.5.2.1. After successful login, key company information will be automatically retrieved and presented to the user based on their role.

4. Assumptions & Constraints

4.1. Assumptions:

4.1.1. It is assumed that administrators will receive the necessary training to manage roles, permissions, and user sessions effectively.

4.1.2. The technology stack will support all the necessary features to manage user roles, access controls, and session tracking.

4.2. Constraints:

4.2.1. Changes to user role requirements or administrative functionalities after the initial implementation phase may lead to delays in project delivery.

4.2.2. The availability of skilled personnel to manage the administration tasks and ensure proper system governance may impact project timelines.

Objective Testing
Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The **Objective Testing Screen** enables users to evaluate models using selected **audit file sets**, **features**, and **score types**.

1.2. Users can **upload files** for evaluation directly through the screen.

1.3. After the evaluation is completed, the screen displays **relevant metrics** based on the evaluation results.

1.4. The interface is designed to **guide users** through the evaluation process.

1.5. The system ensures **accuracy** in user inputs by providing clear instructions and validations.

2. Business Objectives

2.1. The objectives of the **Objective Testing Screen** are:

2.2. To provide a seamless interface for selecting audit file sets, features, and score types for model evaluation.

2.3. To facilitate accurate and efficient file uploads for model evaluation.

2.4. To display evaluation results in the form of metrics after successful completion.

3. Requirements

3.1 Functional Requirements

3.1.1. Audit File Sets:

3.1.1.1. Users must select a single audit file set to be used for evaluation.

3.1.1.2. Only one file set can be selected per evaluation.

3.1.1.3. The interface presents a screen for users to select a single file set, with an option to "Add New Fileset". Upon clicking "Add New Fileset," users are directed to a new screen, where the following options are present:

3.1.1.3.1. They can upload an **Input File,** mandatory

3.1.1.3.2. They can upload an **Output File**, mandatory

3.1.1.3.3. They can upload an optional **Reference File**.

3.1.1.3.4. All files must be in TXT, JSON, WAV format with a maximum size of 5 MB.

3.1.1.3.5. In future implementations, users will have the ability to configure a default file set for each project type and score.

3.1.1.3.6. Future updates will also support handling multiple file sets within a single evaluation.

3.1.2. Feature Selection:

3.1.2.1. Users must select a specific feature to evaluate from the available options, which include:

3.1.2.1.1. Summarization

3.1.2.1.2. Translation

3.1.2.1.3. Transcription

3.1.2.1.4. Q&A

3.1.2.1.5. Text Generation

3.1.3. Score Type:

3.1.3.1. Users must select a score type from a dropdown menu to determine how the model's performance will be evaluated. The available score types are (More will be added in future):

3.1.3.1.1. T**ext Quality-** The aggregate text quality score indicates how well the summary meets criteria: higher scores reflect higher quality. Key factors include grammatical accuracy, non-redundancy, clear references, and coherent structure.

3.1.3.1.2. CoSelection- A "co-selection score" assesses the relevance of elements selected for inclusion in a summary by analyzing topic frequency, conversation flow, engagement level, and sentiment. This score helps prioritize key parts of a call transcript, ensuring concise, informative summaries that capture the main points.

3.1.3.1.3. Completeness- The co-selection score, a proprietary metric, uses entity detection in large language models (LLMs) to compare entities identified in a summarized file with those in an ideal summary. By analyzing the count and content differences between the two

summaries, this score evaluates the completeness and accuracy of the summarized content relative to the ideal summary.

3.1.3.1.4. Utility Score- The Utility Score measures a summary's effectiveness in addressing user queries based on the original content. It checks if the summary contains sufficient, relevant information to accurately answer questions, ensuring it retains the essential utility of the original file.

3.1.3.1.5. No PHI- This Score ensures that the content is free from information that could potentially harm an individual, including sensitive details that might cause physical, emotional, or psychological harm, prioritizing user safety and well-being.

3.1.3.1.6. No PCI- This Score ensures the absence of sensitive financial information, such as bank account or credit card numbers, ensuring the protection of the user's financial privacy and security.

3.1.3.1.7. No PII- This Score ensures that content excludes any personally identifiable information (PII), such as names, addresses, or social security numbers, to safeguard privacy and comply with data protection regulations.

3.1.3.1.8. Absence of Bad Words- This Score ensures that the content is free from offensive, vulgar, or inappropriate language, promoting a respectful and professional tone by filtering out harmful or derogatory terms unsuitable for the intended audience.

3.1.3.1.9. WEC Level- WEC Level (Word Error Count Level) quantifies transcription errors by counting the incorrect words in the transcribed text compared to the original audio, helping to assess the precision and reliability of the transcription process.

3.1.3.1.10. Word Loss Level- The Word Loss Level score measures the number of missing words in a transcribed text compared to the original audio file by calculating the difference in word count, highlighting omissions or errors and helping to evaluate the transcription service's accuracy.

3.1.4. Evaluate Button:

3.1.4.1. Once all required fields (Audit File Sets, Feature, and Score Type) are filled, users can click "Evaluate" to initiate the evaluation process.

3.1.5. Metrics Display:

3.1.5.1. After the evaluation, the system will display the evaluation results as metrics according to the selected score type. For example, if **Absence of Bad Words** is selected, the corresponding metric will reflect the presence or

absence of inappropriate language. Other score types, like **Text Quality** or **No PII**, will present relevant metrics based on the evaluation type chosen.

3.1.5.2. The metrics will be displayed as a bar chart, similar to the visual format shown in the interface, where values for each metric (e.g., **Text Quality**, **CoSelection**, **Completeness**) are represented on a scale from 0 to 10.

3.1.5.3. The color code of the metrics should be (Score < 5 ; Color = red, Score < 8 ; Color = amber, Score >= 8 ; Color = green).

3.1.6. Error Handling:

3.1.6.1. If any required fields are not filled, the system will display an error message, preventing users from proceeding with the evaluation.

3.2. Non-Functional Requirements

3.2.1. Performance:

3.2.1.1. The evaluation process should not exceed xxx seconds after clicking the "Evaluate" button for small models.

3.2.2. Usability:

3.2.2.1. Clear labels and user-friendly interface to guide users through the evaluation process.

3.2.2.2. Validation messages for missing required fields (Audit File Sets, Feature, Score Type).

3.2.3. Reliability:

3.2.3.1. The system should handle multiple evaluations concurrently without performance degradation.

3.2.4. Security:

3.2.4. Ensure that only authorized users can access the evaluation screen and upload files for model evaluation.

3.3. Business Rules

3.3.1. Field Identification and Requirements for Objective Testing Screen

3.3.1.1. Audit File Set (Mandatory):

3.3.1.1.1. Description: Specifies the file set being used for model evaluation.

3.3.1.1.2. UI Validation:

3.3.1.1.2.1. Cannot be left empty.

3.3.1.1.2.2. Users must select a single audit file set for evaluation from predefined options.

3.3.1.1.2.3. Only one file set can be selected per evaluation.

3.3.1.2. Feature Selection (Mandatory):

3.3.1.2.1. Description: Defines the specific feature to be evaluated in the model. Currently available features are:

3.3.1.2.1. Summarization

3.3.1.2.2. Translation

3.3.1.2.3. Transcription

3.3.1.2.4. Q&A

3.3.1.2.5. Text Generation

3.3.1.2.2. UI Validation:

3.3.1.2.2.1. Cannot be left empty.

3.3.1.2.2.2. Must be selected from predefined dropdown options.

3.3.1.2.2.3. One feature per evaluation.

3.3.1.3. Score Type (Mandatory):

3.3.1.3.1. Description: Defines how the model's performance will be evaluated. Current score types include:

3.3.1.3.1.1. Text Quality

3.3.1.3.1.2. CoSelection

3.3.1.3.1.3. Completeness

3.3.1.3.1.4. Utility Score

3.3.1.3.1.5. No PHI

3.3.1.3.1.6. No PCI

3.3.1.3.1.7. No PII

3.3.1.3.1.8. Absence of Bad Words

3.3.1.3.1.9. WEC Level

3.3.1.3.1.10. Word Loss Level

3.3.1.3.2. UI Validation:

3.3.1.3.2.1. Cannot be empty.

3.3.1.3.2.2. Must be selected from a dropdown.

3.3.1.3.2.3. More than one score type can be selected per evaluation.

3.3.2. UI Validation Rules Overview

3.3.2.1. Mandatory Fields:

3.3.2.1.1. Fields such as **Audit File Sets**, **Feature**, **Score Type** must be filled before submission.
3.3.2.1.2. Error messages must be displayed if any mandatory field is left blank or contains invalid input.

3.3.2.2. Data Types:

3.3.2.2.1. Fields such as **Feature** & **Score Type** must be selected from dropdowns with valid options.

3.3.2.3. File Upload Validation:

3.3.2.3.1. Audit File Sets must comply with accepted formats (TXT, JSON, WAV) and file size limits (maximum 5MB).

3.3.3. Evaluation Submission Rules

3.3.3.1. Form Submission:

3.3.3.1.1. The evaluation can only be submitted when all mandatory fields are populated with valid data.

3.3.3.1.2. Error messages should guide users in correcting specific issues before submission.

3.3.3.2. Timeframe for Completion:

3.3.3.2.1. Each evaluation must have a predefined timeframe for completion, and tasks not completed within the timeframe should be flagged for follow-up.

3.3.3.3. Edit and Logging:

3.3.3.3.1. Once submitted, only authorized personnel can edit evaluation details. All changes must be logged for traceability.

4. Assumptions & Constraints

4.1. Assumptions:

4.1.1. Users will have the necessary permission to perform evaluations.

4.1.2. Users will know the type of models they are evaluating and the required files to upload.

4.2. Constraints:

4.2.1. Only one audit file set can be selected at a time for evaluation.

4.2.2. The file upload size for evaluations is limited to TXT files and must be within 5 MB.

Reports Module

Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The Reports Module provides users with detailed insights into evaluations conducted by the system.
1.2. It Includes various report types: historical, diagnostic, predictive, and prescriptive.
1.3. It Assists users in tracking trends, diagnosing issues, and improving model performance.
1.4. It Aims to provide a robust and efficient way to generate, store, and query evaluation data.

2. Business Objectives

The primary objectives of the **Reports Module** are:

2.1. To offer detailed, insightful reports to track and analyze model evaluations.
2.2. To facilitate predictions and diagnostic reporting based on historical data.
2.3. To provide expert recommendations and solutions for improving evaluations.
2.4. To ensure all evaluation data is stored, retrievable, and not deleted to maintain historical integrity.
2.5. Prescription report will be a recommendation to client & is not a self-healing method.

2.6. To generate KPI (Key Performance Indicator) reports that help measure and assess the performance of the evaluations.

3. Requirements

3.1. Functional Requirements

3.1.1. Evaluation Logs:

3.1.1.1. Each evaluation must have a unique reference ID.
3.1.1.2. All evaluations should be stored in a log file (the log structure is hierarchical JSON and managed in SQL).

3.1.2. Types Of Reports:

The Reports Module provides users with various types of reports designed to offer a holistic view of the model evaluation process. These reports enable users to analyze past performance, identify issues, predict future outcomes, and receive expert recommendations. Below is an expanded description of each type of report available in the system:

Each report allows users to apply filter parameters to refine the results. Currently, only two filters are available: *From Date* and *To Date*. However, it is anticipated that additional filter parameters may soon become necessary. Although these additional filters are not currently implemented, the system should be designed to allow for future configuration options, enabling users to apply custom filters as their requirements evolve.

3.1.2.1. Trend Reports:

Trend reports are **historical reports** that allow users to observe how key metrics have changed over time. They provide valuable insights into the performance of models and systems, enabling users to make data-driven decisions. The following are the different types of **Trends Reports** available:

3.1.2.1.1. Accuracy Trend Report:

3.1.2.1.1.1. Purpose: This report calculates the average accuracy score of evaluations on a daily basis for a particular client. It allows users to track the accuracy trends over time and observe how the system's performance is evolving.

3.1.2.1.1.2. How It Works:

3.1.2.1.1.2.1. The system calculates the average accuracy score based on evaluations conducted during a single day.

3.1.2.1.1.2.2. For example, if on **Day 1**, six evaluations were performed (two Algorithmic evaluations with scores of 6 and 7, two Programmatic Interviews with scores of 6 and 7, and two SME audits with scores of 6 and 7), the accuracy trend would be calculated as: Average score = (6+7+6+7+6+7)/6=6.5

3.1.2.1.1.3. Usage: This report is ideal for tracking how the accuracy of evaluations changes over time for a particular client, helping to identify any patterns or shifts in performance.

3.1.2.1.2. Bias Trend Report:

3.1.2.1.2.1. Purpose: This report provides users with a daily average score focusing on **bias evaluations** for a specific client. It gives insights into whether bias is being properly managed and controlled in the system.

3.1.2.1.2.2. How It Works:

3.1.2.1.2.2.1. The system checks if bias evaluations have been performed in the **Programmatic Interview** and **SME audit**. If the data exists, it calculates the average score and generates the report.

3.1.2.1.2.2.2. The report tracks bias performance across various evaluations for a single client.

3.1.2.1.2.3. Usage: Bias trend reports are essential for tracking the system's fairness and ensuring that bias issues are addressed over time.

3.1.2.1.3. Explainability and Safety Trend Reports:

3.1.2.1.3.1. Purpose: These reports work similarly to the **Bias Trend Report** but focus on explainability and safety. They track how well the system is doing in terms of making its evaluations understandable and ensuring they meet safety standards.

3.1.2.1.3.2. How It Works:

3.1.2.1.3.2.1. The system checks for evaluations related to explainability and safety within the **Programmatic Interview** and **SME audit**, calculating the average scores on a daily basis.

3.1.2.1.3.2.1. Usage: These reports help ensure that evaluations meet required safety standards and are understandable to users and stakeholders.

3.1.2.1.4. Client Trend Report:

3.1.2.1.4.1. Purpose: This report provides the average evaluation score for all evaluations conducted in a single day for a particular client.

3.1.2.1.4.2. How It Works:

3.1.2.1.4.2.1. The system aggregates all evaluation scores conducted for a client during a day and calculates the average score to show overall performance.

3.1.2.1.4.3. Usage: Client trend reports are useful for tracking the daily overall performance of evaluations for a specific client, helping to measure client satisfaction and system reliability.

3.1.2.1.5. Feature Trend Report:

3.1.2.1.5.1. Purpose: This report tracks how often a particular feature (such as **Summarization**, **Translation**, or **Text Generation**) is used in the evaluations over a specified period.

3.1.2.1.5.2. How It Works:

3.1.2.1.5.2.1. The system calculates the number of times a particular feature has been used on a daily basis.

3.1.2.1.5.2.2. For example, if **Summarization** was evaluated 5 times on **Day 1** and 7 times on **Day 2**, the feature trend report would show these numbers for the specified time frame.

3.1.2.1.5.3. Usage: Feature trend reports are helpful for tracking which features are being evaluated more frequently, helping teams to allocate resources based on feature popularity or performance.

3.1.2.1.6. Measure Trend Report:

3.1.2.1.6.1. Purpose: This report counts the number of times specific topics have been evaluated on a daily basis. It includes data on **Text Quality** and other key metrics.

3.1.2.1.6.2. How It Works:

3.1.2.1.6.2.1. The system tracks how often certain topics (like Text Quality) are evaluated daily and aggregates this data into a report.

3.1.2.1.6.3. Usage: This report is ideal for measuring how frequently particular topics are being evaluated, helping teams focus on areas that require more attention or analysis.

3.1.2.1.7. Project Trend Report:

3.1.2.1.7.1. Purpose: This report tracks how many times a particular project has been evaluated, using the **Project ID** as the key identifier.

3.1.2.1.7.2. How It Works:

3.1.2.1.7.2.3. The system calculates how many evaluations have been performed for a specific project each day and compiles this data into a trend report.

3.1.2.1.7.3. Usage: Project trend reports help stakeholders monitor the progress of evaluations for a specific project, ensuring that the project is receiving the necessary focus and resources.

3.1.2.1.8. Response Time Trend Report:

3.1.2.1.8.1. Purpose: This report calculates the average time taken for evaluations on a daily basis, allowing users to track how response times fluctuate over time.

3.1.2.1.8.2. How It Works:

3.1.2.1.8.2.1. The system calculates the total time taken for evaluations conducted on a particular day and provides the average evaluation time.

3.1.2.1.8.2.2. For example, if **6 evaluations** were performed on **Day 1**, the system calculates the average time for these evaluations and generates the report.

3.1.2.1.8.3. Usage: Response time trend reports are critical for monitoring system performance and ensuring that evaluations are processed efficiently.

3.1.2.2. Prediction Reports:

3.1.2.2.1. Purpose: These reports build upon the **Trend Reports**, using historical data to forecast future outcomes. They apply statistical or machine learning models to the existing data to predict how the model's performance will evolve.

3.1.2.2.2. Example: After viewing a trend report showing a decline in **PHI scores**, the user may request a prediction report to estimate the score for the next quarter. The system would generate a forecast based on historical data trends.

3.1.2.2.3. Data Source: Like trend reports, prediction reports utilize the evaluation logs but apply predictive algorithms to generate future estimates.

3.1.2.2.4. Usage: Prediction reports are essential for decision-makers who need to plan ahead or adjust their strategies based on anticipated model performance.

3.1.2.3. Diagnostic Reports:

3.1.2.3.1. Purpose: Diagnostic reports are used to analyze and identify problems within the evaluation process. They provide detailed insights into specific failures or underperformance, offering users the ability to drill down into the root cause of issues.

3.1.2.3.2. Example: If a user notices that the **PHI score** has consistently been below a threshold (e.g., 8), they can generate a diagnostic report. This report would highlight the specific models or instances where the score failed, helping the user to pinpoint the cause of the issue.

3.1.2.3.3. Triggers: Diagnostic reports can be automatically triggered when a specific condition is met, such as a score falling below a set threshold.

3.1.2.3.4. Usage: These reports are critical for troubleshooting and addressing specific problems in the evaluation process, enabling quicker identification and resolution of issues.

3.1.2.4. Prescription Reports: Prescription BRD.docx

3.1.2.4.1. Purpose: Prescription reports go beyond identifying problems and provide actionable solutions to improve the evaluation process. This module will be utilized in the Reports Module as well as the SME Portal. Utilizing AI model (RAG model), the reports help SME identify areas of concern and recommend corrective actions.

3.1.2.4.2. How It Works:

3.1.2.4.2.1 The SME module uses an RAG model to indicate areas of concern. A red flag signals a critical issue, amber highlights areas that need attention, and green denotes satisfactory performance.

3.1.2.4.2.2. The **"Improve"** button in the SME module allows users to take corrective actions. It has two functionalities:

3.1.2.4.2.2. Discuss:

3.1.2.4.2.2.1. The AI models collaborate to generate potential solutions for improving performance. The AI can analyze historical data, trends, and even input from other models to recommend changes.

3.1.2.4.2.2.2. Here there will be 3 options of "Chat", "Call" & "Whiteboard".

3.1.2.4.2.2.3. Chat- It a normal chat conversation feature where we can "Start" and "Stop" the chat. We can have a chat with the AI models and discuss with them as Human Beings. And we can query and get their opinions

and all this stuff and collect the responses and deliver it as a recommendation.

3.1.2.4.2.2.4. Call- It a normal audio conversation feature where we can “Start” and “Stop” the call. It is just an audio conversation with the AI models.

3.1.2.4.2.2.5. Whiteboard- The idea is essentially just as we as human beings are able to talk with each other and chat and if there is a whiteboard, you can work on that whiteboard together. The same we are able to do here with these AI models which are RAG models.

3.1.2.4.2.3. Auto Optimize:

3.1.2.4.2.3.1. This is prompt refinement. Here we provide the Prompt and Objective & get the best version of the prompt.

3.1.2.4.2.3.2. The AI will take the provided prompt & objective and convert it into the best version of the prompt.

3.1.2.4.2.3.3. SME can verify the prompt and provide that as SME feedback.

3.1.2.4.3. Example: A user might ask, "Why has the **PHI score** been consistently low for the last 2 years?" The prescription report would analyze the data and suggest specific changes or optimizations, such as adjusting model parameters or altering the input data.

3.1.2.4.4. Usage: These reports are highly valuable for continuous improvement, allowing users to not only identify issues but also receive expert guidance on how to address them.

3.1.2.5. Ticker Reports:

3.1.2.5.1. Purpose: Ticker reports are live, real-time updates that appear as a ticker on the homepage or other prominent areas of the application. They provide users with up-to-the-minute information on evaluations or any critical updates.

3.1.2.5.2. How It Works: The Ticker can be project-based or organization based. The ticker is powered by a report API that fetches

data in real-time, ensuring that users always have access to the most current information without having to manually generate a report.

3.1.2.5.3. Example: A ticker might show the **number of evaluations** completed in the last hour or display alerts when a critical score drops below a certain threshold.

3.1.2.5.4. Usage: Ticker reports are essential for users who need to monitor live evaluation data without manually generating a full report.

3.1.2.6. Summary Reports:

3.1.2.6.1. Purpose: These are high-level reports that provide an overview of the evaluation process, summarizing key metrics and results. Summary reports give users a bird's-eye view of the system's performance over a specified period or for a particular evaluation set.

3.1.2.6.2. Example: A user can generate a **monthly summary report** that aggregates the most critical metrics such as PHI scores, compliance rates, and the number of evaluations completed. The report will provide an easy-to-read summary of all relevant data.

3.1.2.6.3. Data Source: These reports compile data from multiple evaluations and provide a condensed version for easy consumption.

3.1.2.6.4. Usage: Summary reports are ideal for executives or stakeholders who need a quick and comprehensive overview of the system's performance without going into the technical details.

3.1.2.7. KPI Reports:

3.1.2.7.1. Purpose: Key Performance Indicator (KPI) reports track specific metrics that measure the overall health and effectiveness of the evaluation process. These reports provide insights into whether evaluations are meeting their objectives and how well the system is functioning.

3.1.2.7.2. Example: A **KPI report** might show how many evaluations were completed on time, whether all required metrics were covered, and the average response time for evaluations.

3.1.2.7.3. Metrics Monitored:

3.1.2.7.3.1. Evaluation Coverage: Ensures that all required metrics (such as Toxicity, Compliance, Efficacy, etc.) are evaluated.

3.1.2.7.3.2. Response Time: Tracks how quickly evaluations are processed and whether they are meeting performance targets.

3.1.2.7.4. Usage: KPI reports are used to monitor system efficiency and ensure that all critical aspects of the evaluations are being tracked and optimized.

3.1.2.7.5. Error Handling:

3.1.2.7.5.1. The system must generate an error if report generation fails due to missing or invalid data.

3.1.2.8. Reports Output: **There should be a section called report output that describes the way the report will be rendered to the user. The current dev UI is very underdeveloped. It just shows a table and maybe a line chart at the bottom. In the new react UI this needs to be detailed out**

3.2. Non-Functional Requirements

3.2.1. Performance: Report generation should not exceed xxx **seconds** for smaller datasets.

3.2.2. Scalability: The system should support concurrent report generation without degradation in performance.

3.2.3. Usability: The UI should guide users through the report generation process with clear labels and validation prompts.

3.2.4. Security: Only authorized users should be allowed to generate or view reports.

3.2.5. Data Integrity: No evaluation data should be deleted from the system to preserve historical records.

3.3. Business Rules

3.3.1. Evaluation Storage: All evaluations must be stored and never deleted, as they represent the execution history.

3.3.2. Report Generation: Reports must be generated based on the data stored in hierarchical JSON format in SQL.

3.3.3. Access Control: Only authorized personnel can generate or access specific types of reports (e.g., Prescription).

3.3.4. KPI Reports: The system must generate KPI (Key Performance Indicator) reports to measure and assess evaluation performance.

4. Assumptions & Constraints

4.1. Assumptions

4.1.1. Users generating reports will have the necessary permissions.
4.1.2. All necessary data for generating reports will be available in the SQL database.

4.2. Constraints

4.2.1. Report queries may be more complex due to the hierarchical JSON format.
4.2.2. The size of the evaluations data could affect report generation time for large datasets.
4.2.3. Reports are limited to evaluations stored in the system's log.

Evaluation
Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The **Evaluation module** allows various users at different levels to perform specific evaluations.

1.2. The system supports four types of users:

1.2.1. C-level users: Perform ad-hoc & UI based evaluations.

1.2.2. Testing users: Conduct sprint-level evaluations.

1.2.3. Developers: Perform internal workflow evaluations.

1.2.4. Scheduled evaluations are also supported for routine assessments.

1.3. There are three types of evaluations:

1.3.1. Objective evaluations: Fast, synchronous evaluations with quick responses.

1.3.2. Reflective evaluations: AI-driven, asynchronous interviews that take longer to complete.

1.3.3. Investigative evaluations: Human-led, asynchronous evaluations with longer response times.

1.4. The system aims to:

1.4.1. Streamline and automate evaluations.

1.4.2. Offer real-time monitoring of evaluation progress.

1.4.3. Provide alerts for critical situations or specific thresholds.

2. Business Objectives

2.1. Provide a flexible evaluation platform for different user roles.

2.2. Enable multiple evaluation types: Objective Interview (synchronous), Reflective Interview (asynchronous), and Investigative Testing (SME) Interview (human-based, asynchronous).
2.3. Ability to add customized evaluations.
2.4. Allow scheduling of evaluations to ensure timely feedback.
2.5. Implement a real-time alert system to monitor scores and send notifications when thresholds are breached.

3. Requirements

3.1. Functional Requirements

3.1.1. Ad-hoc & UI Based Evaluation (C-level users): C-level users should be able to perform immediate evaluations without predefined schedules.
3.1.2. Sprint-level Evaluation (Testing users): Testing users should be able to evaluate specific sprints in the development cycle.
3.1.3. Internal Workflow Evaluation (Developer level): Developers will be able to perform evaluations within the workflow to ensure smooth functionality.
3.1.4. Scheduled Evaluation: All types of evaluations can be scheduled (e.g., weekly on a specific day).
3.1.5. Evaluation Types:

3.1.5.1. Objective Testing: Objective Testing BRD.docx
Fast, synchronous, and automated evaluation that provides real-time feedback.
3.1.5.2. Reflective Testing: Reflective Testing BRD
Asynchronous process that takes a longer time.
3.1.5.3. Investigative: Investigative BRD.docx
Human-based evaluation that is also asynchronous and requires longer processing times.

3.1.6. Alert Message: The system will provide an alert message to send notifications if an evaluation score falls below a defined threshold, supporting real-time monitoring.

3.2. Non-Functional Requirements

3.2.1. Scalability: The system should handle a growing number of evaluations without impacting performance.

3.2.2. Performance: Algorithmic evaluations should provide instant feedback, while Interview and SME evaluations can handle longer response times.
3.2.3. Usability: The interface should be intuitive for different user levels to perform evaluations efficiently.
3.2.4. Reliability: Scheduled evaluations must trigger without fail, and alerts should be sent promptly.

3.3. Business Rules

3.3.1. All users should be able to do all kinds of evaluations.

3.3.2. Based on the priorities:

3.3.2.1. C-level users should be able to perform ad-hoc evaluations & UI based evaluations only.

3.3.2.2. Testers can do the UI based evaluations but mostly the sprint level evaluations.

3.3.2.3. Developers are able to do internal workflow evaluations, i.e., nuance customized API driven evaluations.

3.3.3. All users can schedule evaluations according to their evaluation type.

3.3.4. Alerts will be sent automatically when an evaluation score falls below a predefined threshold.

4. Assumptions & Constraints

4.1. Assumptions:

4.1.1. The evaluation types and user access rights are pre-defined and will not change frequently.
4.1.2. Users will follow the specific role-based restrictions for evaluations.
4.1.3. The system will have enough resources to process the asynchronous evaluations without delays.

4.2. Constraints:

4.2.1. Response times for asynchronous evaluations (Interview and SME) may vary based on the complexity and workload.
4.2.2. Real-time monitoring will depend on the performance of the underlying alerting infrastructure.

4.2.3. Scheduled evaluations will rely on the system's internal scheduler, which must be consistently available.

Business Requirements Document (BRD) Project Assessment

1. Executive Summary

1.1. The Project Assessment module enables users to create, read, update, and delete projects and AI models.

1.2. It provides functionality to assess threats and opportunities related to AI projects.

1.3. Helps ensure that potential risks are identified and mitigated, while opportunities are leveraged.

1.4. Aims to assess, evaluate, and streamline the AI impact on organization-wide projects.

1.5. Allows users to effectively monitor AI performance effectively across various projects.

2. Business Objectives

2.1. Efficient Project Management: Provide users with a robust interface to evaluate the AI footprint across multiple projects, offering complete functionality for assessing the impact and performance of AI models.

2.2. Risk & Opportunity Evaluation: Ensure that users can easily assess the risks and opportunities associated with each AI model in a project, enabling proactive adjustments to maximize the benefits of AI implementation.

2.3. Data-Driven Decision-Making: Enable users to make informed decisions based on comprehensive risk assessments and the identification of opportunities for each AI model, leading to optimized outcomes across projects.

3. Requirements

3.1. Functional Requirements

3.1.1. CRUD Operations for Projects and AI Models

3.1.1.1. Add: The system must allow users to create new projects and associate them with AI models. Users should be able to define project-related fields (name, description, AI models, Type of Model, Features).

3.1.1.1.1. Create Project:

3.1.1.1.1.1. Project Name- Name of the project
3.1.1.1.1.2. Description- short description of the project.

3.1.1.1.2. Create Model:

3.1.1.1.2.1. Model Name- Azure AI, Google AI, Llama2, HuggingFace, Open Ai
3.1.1.1.2.2. Type of use: Select the type of model from a dropdown list (required):

3.1.1.1.2.2.1. Foundational Model- A **Foundational Model** in AI refers to a large-scale machine learning model trained on vast amounts of diverse data,

typically in an unsupervised or self-supervised manner.

3.1.1.1.2.2.2. Foundational Model with RAG- A **Foundational Model with RAG** refers to the combination of a **Foundational Model** and the **Retrieval-Augmented Generation (RAG)** technique.

3.1.1.1.2.2.3. Foundational Model with Agents- A **Foundational Model with Agents** refers to integrating a foundational AI model with specialized autonomous systems, or **agents**, that can perform specific tasks or actions based on the model's outputs.

3.1.1.1.2.2.4. Custom Model- A **Custom Model** in AI refers to a machine learning model specifically designed and trained for a particular task, domain, or dataset.

3.1.1.1.2.2.5. Models with Evaluation- Models with Evaluation refer to AI or machine learning models that are assessed using various metrics and methods to determine their performance and reliability on specific tasks.

3.1.1.1.2.2.6. LLM with RAFT- An **LLM with RAFT** refers to combining a **Large Language Model (LLM)** with the **RAFT** consensus algorithm, commonly used in distributed systems to manage leader election and maintain consistency across multiple nodes.

3.1.1.1.2.3. Features: User can select multiple Features from dropdown.

3.1.1.1.2.3.1. Summarization- The process of condensing a larger text into a shorter version while retaining its key information and meaning.

3.1.1.1.2.3.2. Translation- Converting text or speech from one language into another while preserving the original meaning.

3.1.1.1.2.3.3. Transcription- The conversion of spoken language or audio content into written or text format.

3.1.1.1.2.3.4. Q&A- An AI task where the system responds to questions based on given data or context, providing accurate and relevant answers.

3.1.1.1.2.3.5. Text Generation- The process of creating coherent and contextually appropriate text content using AI, often based on a given prompt or input.

3.1.1.2. Read: Users must be able to view existing projects (By ID & Name) and AI models, including all associated metadata, configuration details, scores& interview details.

3.1.1.2.1. Projects:

3.1.1.2.1.1. Project ID: Unique ID. Can search with this ID.
3.1.1.2.1.2. Project Name: Unique name of the project. Can search with the name.
3.1.1.2.1.3. Description: Description of the project.
3.1.1.2.1.4. Status: Active/Inactive
3.1.1.2.1.5. Created Date: Project creation date (GMT) (Sorting based on Created date)
3.1.1.2.1.6. Created By: Name of the person who created the Project.

3.1.1.2.2. Models (Inside specific Project)
3.1.1.2.2.1. Model ID: Unique ID can search with this id.
3.1.1.2.2.2. Model name: Name of the model. Can be used for searching.
3.1.1.2.2.3. Type: Type of the model.
3.1.1.2.2.4. Created date: Creation date of the model. (GMT) (Sorting based on Created date)
3.1.1.2.2.5. Last Evaluation: Evaluation status.
3.1.1.2.2.6. Status: Status of the model.

3.1.1.2.3. Pagination: Maximum 5 projects/modules within a page.

3.1.1.3. Update: Users should be able to modify existing projects and AI models, such as modifying AI model features.
3.1.1.3.1. Project List: Status & ?
3.1.1.3.2. Module: We can modify listed fields.
3.1.1.3.2.1. Model name
3.1.1.3.2.2. Type
3.1.1.3.2.3. Last Evaluation
3.1.1.3.2.4. Status

3.1.1.4. Delete: The system must support the deletion of projects and AI models, ensuring that all associated data is properly managed and archived if necessary.
3.1.1.4.1. Project:
3.1.1.4.1.1. ID & Name: can delete by id & name
3.1.1.4.2. Model:
3.1.1.4.1.2. ID: can delete by ID.

3.1.2. Key Features:

3.1.2.1. The system should log all CRUD actions for audit purposes.
3.1.2.2. Error handling should be in place to handle cases where invalid or missing data is entered.

3.2. Threat and Opportunity Assessment

3.2.1. Assessment Categories: The system evaluates AI projects at the **organization level** based on the following key areas:

3.2.1.1. Toxicity: Measures the potential harmful effects or biases present in the AI models. A lower score (0-10) indicates higher toxicity, while a higher score reflects a safer, less biased model.

3.2.1.2. Compliance: Assesses the AI models' adherence to industry regulations, legal standards, and ethical guidelines. A score between 0-10 reflects the degree of compliance.
3.2.1.3. Efficacy: Evaluates the effectiveness and accuracy of AI models in meeting objectives and producing desired outcomes. A higher score indicates greater efficacy.
3.2.1.4. Maturity: Determines the development and stability of the AI models, considering factors such as the robustness of algorithms, data inputs, and consistency of outputs.

3.2.2. Scoring System:

3.2.2.1. Each assessment category is scored between **0-10** at the **project level**, where:

3.2.2.1.1. 0-6 indicates critical risks or underperformance, deeming it **not safe**.
3.2.2.1.2. Scores greater than **6** reflect strong performance or minimal risk.

3.2.2.2. At the **organization level**, the scores are averaged across all projects to provide an overall assessment of AI initiatives.

3.2.3. Recommendations:

3.2.3.1. Based on the average assessment scores, the system generates **organization-level recommendations** for addressing high-risk areas (e.g., reducing toxicity or improving compliance) and maximizing strengths (e.g., enhancing efficacy or maturity).

3.2.3.2. While project-level assessments are not yet presented in the UI, they are implemented in the API. Overall, org-wide assessment is calculated by averaging project-wise assessments, and eventually, this will allow for detailed recommendations specific to each individual project.

3.2.4. Key Features:

3.2.4.1. Risk assessments are **dynamically updated** based on project progress and AI model iterations.
3.2.4.2. Users can view a **detailed breakdown** of each category score along with specific recommendations for improvement.
3.2.4.3. Assessment reports are exportable in PDF and other formats for review by authorized users.

4.2. Non-Functional Requirements

4.2.1. Performance: Assessments should be calculated and displayed in real-time based on project data, ensuring that users receive up-to-date feedback.

4.2.2. Security: Only users with the appropriate permissions can view and modify assessment data, especially for sensitive projects.

4.3. Business Rules

4.3.1. Unique Identification and Mandatory Details: Each project must have a unique Project ID, with mandatory name and description, and models must have a unique Model ID.

4.3.2. Required Fields and Validation: Model Name, Type, and Feature fields are required, and CRUD operations must follow standard validation rules (e.g., required fields, character limits).

4.3.3. Permissions: Only users with appropriate permissions can modify projects and models.

4.3.4. Standardized Evaluation and Data Retention: Toxicity, Compliance, Efficacy, and Maturity must be evaluated consistently, and historical scores must be stored for auditing and tracking.

5. Assumptions & Constraints

5.1. Assumptions:

5.1.1. Users will have the necessary permissions to manage projects and perform assessments.
5.1.2. Real-time data inputs will be available for risk and opportunity analysis.

5.2. Constraints:

5.2.1. System performance may be impacted if large volumes of data are being processed simultaneously during assessments.
5.2.2. Security protocols and data protection laws must be adhered to when managing and accessing sensitive project information.

Prescription

Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The Prescription Reports module provides actionable solutions to improve the evaluation process.

1.2. It is integrated into both the Reports Module and the SME Portal.

1.3. Utilizes an AI model (RAG model) to help SMEs identify areas of concern and recommend corrective actions.

1.4. Interactive features, including chat, call, and whiteboard, enable seamless collaboration between SMEs and AI models.

2. Business Objectives

2.1. Enhance Problem-Solving Capabilities: Develop reports that go beyond traditional data analysis by providing actionable insights and specific recommendations to address evaluation challenges.

2.2. Streamline the Evaluation Process: Facilitate the identification of critical issues, prioritize corrective actions, and track improvements, leading to more effective evaluation outcomes.

2.3. Empower Decision-Making: Enable SME to collaborate with AI to discuss and trace solutions, improving their decision-making process by leveraging expert recommendations and data-driven insights.

3. Requirements

3.1. Functional Requirements

3.1.1. Prescription Report Generation:

3.1.1.1. The system must generate solutions that highlight identified problems and provide actionable solutions.

3.1.1.2. Reports should include a summary of findings, recommended actions, and visual indicators using the AI Model (RAG model).

3.1.1.3. The report functionality must exist as a chatbot that can answer user queries related to identified problems and recommended solutions.

3.1.2. Improve Button Functionalities:

3.1.2.1. Discuss Functionality:

3.1.2.1.1. When performance issues are identified, SME can use the Discuss functionality to facilitate dialogue and explore potential solutions in a collaborative manner with the help of AI model (RAG).

3.1.2.1.2. The AI models collaborate with SME to generate potential solutions for improving performance based on interaction with the SME.

3.1.2.1.3. **Chat Option:**

3.1.2.1.3.1. SME can engage in a chat conversation with AI models to query, discuss, collect responses, gather opinions and provide recommendations as feedback.

3.1.2.1.3.2. Start and Stop functions allow SME to control the chat duration.

3.1.2.1.4. **Call Option:**

3.1.2.1.4.1. Allows SME to have a recorded audio conversation with AI models (RAG Model), facilitating a more direct and immediate discussion of solutions.

3.1.2.1.4.2. Start and Stop controls manage the call session.

3.1.2.1.5. **Whiteboard Option:**

3.1.2.1.5.1. Provides a collaborative visual space where SME and AI models (RAG Model) can work together, sharing and drawing ideas in real-time.

3.1.2.1.5.2. Allows annotations, file inputs, diagrams, and live edits, simulating a brainstorming session with AI.

3.1.2.2. Auto Optimize Functionality:

3.1.2.2.1. Auto Optimize is an AI-driven feature that refines prompts to create the most effective version aligned with a specific objective. It enhances the clarity and precision of prompts, ensuring they are clear, actionable, and better suited for generating accurate AI recommendations.

3.1.2.2.2. How It Works:

3.1.2.2.2.1. Input Prompt and Objective: SME provides a prompt along with its intended objective, guiding the AI to understand what needs to be achieved.

3.1.2.2.2.2. AI-Driven Refinement: The AI analyzes and refines the prompt, improving its structure and clarity to better communicate the objective.

3.1.2.2.2.3. Output of the Best Version: The optimized prompt is presented, offering a polished, targeted version that aligns with the goal.

3.1.2.2.2.4. SME Verification and Feedback: The refined prompt is reviewed by an SME, who verifies its relevance and provides feedback, ensuring it meets industry standards.

3.2. Non-Functional Requirements

3.2.1. Performance:

3.2.1.1. Reports must be generated in under x seconds to ensure a smooth user experience.

3.2.1.2. System responsiveness for chat, call, and whiteboard interactions should not exceed a latency of x seconds.

3.2.2. Usability:

3.2.2.1. The interface should be intuitive, with clear guidance for using Discuss and Trace functionalities.

3.2.2.2. Tutorials and help sections should be available to assist SME unfamiliar with AI interactions.

3.2.3. Scalability:

3.2.3.1. The system should support multiple SME generating reports and interacting with AI models simultaneously without degradation in performance.

3.2.4. Security:

3.2.4.1. Data transmitted during interactions (chat, call, whiteboard) must be encrypted to ensure confidentiality.

3.2.4.2. Access to prescription reports and AI models must be restricted to authorized personnel.

3.2.5. Reliability:

3.2.5.1. The system should have x% uptime to ensure availability for critical decision-making processes.

3.2.5.2. Backup mechanisms should be in place to save progress in chat, call, and whiteboard sessions in case of an unexpected interruption.

3.3. Business Rules

3.3.1. Corrective Action Validation:

3.3.1.1. All recommendations provided by the AI must align with internal business policies and any applicable industry regulations.

3.3.1.2. Recommendations should be flagged for review by an SME before implementation if they involve high-risk changes.

3.3.2. Data Privacy:

3.3.2.1. SME must consent to data usage terms, especially when their interactions (chat, call, whiteboard) are recorded for quality and training purposes.

3.3.2.2. AI models should anonymize data inputs where possible to protect sensitive information.

4. Assumptions & Constraints

4.1. Assumptions:

4.1.1. The AI models are regularly updated with relevant data to provide accurate and timely recommendations.

4.1.2. SME have a basic understanding of how to interact with AI tools (chat, call, whiteboard).

4.2. Constraints:

4.2.1. System performance may be affected by the complexity and size of the data sets being analyzed.

4.2.2. The effectiveness of AI-generated recommendations is limited by the quality and availability of historical data.

Reflective Testing

Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. Reflective Testing is a key phase in evaluating AI models, ensuring their accuracy and regulatory standards.

1.2. It tests AI performance across key parameters like accuracy, ethics, fairness, and compliance, ensuring reliable and safe decision-making.

1.3. Results are presented as metrics, providing insights into the model's strengths and areas for improvement.

1.4. This process is critical for verifying the AI's technical reliability and ethical soundness prior to its implementation in practical applications.

1.5. This process involves using AI to evaluate other AI models, helping to check their performance, accuracy, and compliance.

2. Business Objectives

2.1. AI-Driven AI Model Assessment:

2.1.1. This system is uniquely designed for *AI to evaluate AI*, with interviews, questionnaires, scoring, and summarization entirely created and calibrated by AI models, ensuring zero human involvement in the evaluation process.

2.2. Streamlined AI Model Assessments:

2.2.1. Provides a user-friendly interface to initiate and conduct automated assessments of AI models' capabilities, with all evaluation steps performed by AI models themselves.

2.3. Customization of Interviews:

2.3.1. Users can configure key parameters like topics (bias, safety, ethics), evaluation modes (project-specific or core model), and question types (multiple-choice, fill-in-the-blank).

2.4. Detailed Evaluations:

2.4.1. Enables detailed and focused model evaluations tailored to specific needs and contexts.

2.5. Visual Presentation of Results:

2.5.1. Displays evaluation results using clear metrics and visualizations, such as charts and graphs, making it easier to analyze outcomes.

2.6. Compliance with Standards:

2.6.1. Helps ensure AI models meet diagnostic accuracy, subject matter expertise, and regulatory standards.

2.7. Future Adaptability:

2.7.1. Designed for flexibility to accommodate additional topics and question types in future updates.

3. Requirements

3.1. Functional Requirements

3.1.1. Interview Creation:

3.1.1.1. Users must be able to initiate the creation of a programmatic interview by clicking the **Create Interview** button.

3.1.1.2. The system will display a form for interview configuration with the following fields:

3.1.1.2.1. Topic: A required field where users select the topic for the interview. More topics will be added later on. Available topics include:

3.1.1.2.1.1. Bias- A systematic unfairness in AI models where certain groups or outcomes are favored due to imbalanced data or algorithmic design.

3.1.1.2.1.2. Safety- Measures and protocols in AI development that ensure systems operate reliably and do not cause harm to users or environments.

3.1.1.2.1.3. Ethics- The moral principles guiding AI development and deployment, ensuring that AI technologies are used responsibly, fairly, and for the benefit of all.

3.1.1.2.1.4. Explainability- The ability of an AI model to provide clear, understandable reasoning behind its decisions or predictions.

3.1.1.2.1.5. Hallucination- A phenomenon in AI where the model generates incorrect or nonsensical information that wasn't present in the input data.

3.1.1.2.2. Evaluation Mode: A required dropdown where users select the evaluation method:

3.1.1.2.2.1. Project Specific: For evaluations tailored to specific projects, requiring domain tags.

3.1.1.2.2.2. Core Model: Evaluates the core AI model without any domain-specific association (no domain tags required).

3.1.1.2.3. Domain Tag(s): Multi-select field that depends on the topic and evaluation mode chosen (more domain tags will be available in the future):

3.1.1.2.3.1. Insurance

3.1.1.2.3.2. Finance

3.1.1.2.3.3. Information Technology

3.1.1.2.4. Interviewer Model: A required search field where users can select or search for a predefined AI model to conduct the interview. (Request model functionality will be added in future, by

which user can request for a model which is currently unavailable, by filling a simple form)

3.1.1.2.5. Question Type: A multi-select field where users can choose one or more question types. Available options:

3.1.1.2.5.1. Multiple Choice Questions

3.1.1.2.5.2. Fill in the Blanks

3.1.1.2.6. Number of Questions: A numeric input where users specify how many questions should be included in the interview.

3.1.1.2. Form Validation:

3.1.1.2.1. All fields marked with an asterisk (*) must be completed before the form can be submitted.

3.1.1.2.2. The **Domain Tag(s)** field will dynamically update based on the **Topic** and **Evaluation Mode** selected.

3.1.1.2.2.1. When **Project Specific** is chosen, users must select one or more domain tags.

3.1.1.2.2.2. When **Core Model** is selected, the **Domain Tag(s)** field is disabled or marked as "Not Applicable."

3.1.1.2.3. The **Create Interview** button becomes active once all required fields are filled. Upon submission, the system will validate inputs and create the interview for further processing.

3.1.1.3. Future Implementation:

3.1.1.3.1. Multiple Interviewer Models: The system will support multiple AI interviewer models within a single evaluation, providing diverse perspectives and a more comprehensive assessment.

3.1.1.3.2. Multi-Shot Interviews: The system will enable multi-shot interviews, particularly valuable for assessing a model's reasoning capabilities or explainability. In these interviews, each question will initiate a series of steps, allowing the model to explain its reasoning in arriving at an answer. This approach is useful for complex or scientific subjects, where simply providing a correct answer is not sufficient;

instead, the model must demonstrate an understanding of the concepts by showing the reasoning behind its responses.

3.1.2. Interview List Display:

3.1.2.1. After creating an interview, it will appear in the **Interview List** with the following columns:

3.1.2.1.1. Id: A unique identifier for each interview (e.g., maRBHjd).
3.1.2.1.2. Interview Model: The model used for the interview (e.g., OpenAi-Llama-Bias).
3.1.2.1.3. Topic: The selected topic of the interview (e.g., Bias).
3.1.2.1.4. Evaluation Mode: Indicates whether the evaluation is **Project Specific** or **Core Model** (e.g., Project Specific).
3.1.2.1.5. Domain: Displays selected domain tags for **Project Specific** interviews (e.g., Insurance, Finance). For **Core Model** evaluations, this field is left blank.
3.1.2.1.6. # Questions: Number of questions configured for the interview (initially shows "-" if unspecified).
3.1.2.1.7. Question Type: The type of questions used in the interview (e.g., Multiple Choice Questions).

3.1.2.2. Score: Shows the score of the interview once it's completed (it shows "NAN%" if there is an API issue). Otherwise, it shows the complete score in percentage ("83%").
3.1.2.3. Created Date: The date and time the interview was created (e.g., 17-Sept-2024 09:14 AM).
3.1.2.4. Users can navigate through the list using **Previous** and **Next** buttons to view more interviews and initiate new interviews by clicking the **Create Interview** button.

3.1.3. Interview Submission and Results

3.1.3.1. Once the form is submitted, the system will validate the inputs and create the interview. After the interview is completed, the results will be automatically displayed in the **Metrics** section, which includes:
3.1.3.2. Metrics Visualization: Results will be presented through charts, graphs, and tables, allowing users to interpret results easily.
3.1.3.3. Data can also be exported for further analysis and reporting. The

color code of the metrics should be (Score < 5 ; Color = red, Score < 8 ; Color = amber, Score >= 8 ; Color = green).

3.2. Non-Functional Requirements

3.2.1. Usability: The interface should be user-friendly, with clear labels and helpful tooltips for all form fields, and a progress bar should be displayed during lengthy interview processes to improve user experience and responsiveness.

3.2.2. Performance: The system must validate and process interview creation requests in real-time, with interviews conducted asynchronously to handle lengthy evaluations smoothly.

3.2.3. Scalability: The system should support concurrent interview creations by multiple users without performance degradation, ensuring a responsive experience even with high demand.

3.3. Business Rules

3.3.1. Predefined Topics:

3.3.1.1 Interviews can only be created for topics that have been predefined in the system.
3.3.1.2 Only one topic can be selected per interview, and it is a **required field**.
3.3.1.3 Current available topics include:
- **3.3.1.3.1** Bias
- **3.3.1.3.2** Safety
- **3.3.1.3.3** Ethics
- **3.3.1.3.4** Explainability
- **3.3.1.3.5** Hallucination

3.3.2. Project-Specific Interviews:

3.3.2.1 For **Project-Specific** evaluations, users **must** select at least one Domain Tag.
3.3.2.2 Domain Tags relate to the specific industry or domain being evaluated and may include:
- **3.3.2.2.1** Insurance
- **3.3.2.2.2** Finance
- **3.3.2.2.3** Information Technology

3.3.2.3 More domain tags will be added in future updates.

3.3.3. Core Model Interviews:

3.3.3.1 When selecting the **Core Model** evaluation mode, domain tags are **not required**.
3.3.3.2 The Domain Tag(s) field will be marked as **"Not Applicable"** when this mode is selected.

3.3.4. Interview Creation:

3.3.4.1 All required fields marked with an asterisk (*) must be filled before the interview form can be submitted.
3.3.4.2 The system will dynamically update available options in certain fields based on prior selections, like Topics and Evaluation Mode.

3.3.5. Evaluation Mode Constraints:

3.3.5.1. Two evaluation modes are available:
3.3.5.1.1 Project-Specific: requires domain tags.
3.3.5.1.2 Core Model: does not require domain tags.

3.3.6. Validation and Submission:

3.3.6.1 1The interview creation form cannot be submitted until all required fields are completed and inputs pass validation checks.

3.3.7. Requesting Models:

3.3.7.1 If the required AI model is unavailable for conducting the interview, users can request that model by submitting a request form (to be added in future updates).

4. Assumptions & Constraints

4.1. Assumptions:

4.1.1. Users will have prior knowledge of the AI models and their use cases.

4.1.2. System will be regularly updated to include new topics, domains, and question types based on user feedback.

4.2. Constraints:

4.2.1. The system must adhere to existing data security and privacy regulations, especially when handling sensitive information during the interview process.

4.2.2. Performance metrics should be calculated and displayed in the **Metrics** section shortly after the interview is completed.

STUDIO Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. The Studio module of the magicAUDIT application empowers users to create and manage evaluations beyond standard packages.
1.2. Users can submit requests for new types of custom object, offering more flexible assessment capabilities.
1.3. The module supports three main types of requests: Request for scores, Interviewer Models, and Reports.
1.4. This document outlines the business objectives, detailed requirements, and rules governing the Studio module.
1.5. The goal is to ensure successful implementation and alignment with organizational objectives.

2. Business Objectives

2.1. Enhance User Capabilities: Enable users to create customized objects such as scores, interviewer models, and reports tailored to their specific needs.
2.2. Improve Efficiency: Streamline the request process for new evaluations and models, reducing turnaround time from request to availability.
2.3. Leverage AI Technology: Implement an autobuild feature that uses AI to automate code generation for tests, improving responsiveness and minimizing manual intervention.
2.4. Facilitate Reporting: Provide robust reporting capabilities to generate actionable insights through KPIs and other metrics.
2.5. Support User Management: Establish clear administrative controls for overseeing user requests and maintaining the integrity of the evaluation process.

3. Requirements

3.1 Functional Requirements

3.1.1 Evaluation Creation (Test)

3.1.1.1. Create New Evaluation:

3.1.1.1.1. Users shall be able to initiate the creation of a new evaluation by entering a test description (e.g., “Checks clarity of Gen-AI based images”) and specifying the test inputs and will get expected outputs.

3.1.1.1.2. The system shall validate that all required fields (test description, inputs, expected outputs) are filled out before the request is submitted.

3.1.1.2. Request Submission:

3.1.1.2.1. Once the evaluation creation form is completed, users shall be able to submit the request.

3.1.1.2.2. The request shall be sent to the Dev team for manual creation of the test. The Dev team will develop the test based on the provided inputs and plug it into the system for use.

3.1.1.2.3. The system shall provide confirmation to users that the request has been successfully submitted.

3.1.1.2.4. The system shall generate a unique request ID for tracking purposes and display it to the user.

3.1.1.3. Request Management:

3.1.1.3.1. Users shall be able to view the status of all submitted requests in a dashboard, including details such as the request ID, type, submission date, and status (e.g., “Pending,” “Built”).

3.1.1.3.2. Admins shall have the capability to view, approve, or reject evaluation requests based on predefined criteria.

3.1.1.3.3. Once a request is approved, the status shall change to “In Progress” or “Built” depending on the stage of completion.

3.1.2 Autobuild Feature

3.1.2.1. AI-driven Code Generation:

3.1.2.1.1. The system shall allow users to choose between manual request submission or autobuild (AI-generated) options during the evaluation creation process.

3.1.2.1.2. When autobuild is selected, the system shall automatically generate the required test with help of AI and notify the user when it is ready for testing.

3.1.2.1.3. The system shall be capable of generating tests and deploying it within 2 minutes using the autobuild feature.

3.1.2.2. Test Review and Deployment:

3.1.2.2.1. In the future implementation, users shall be able to review the auto-generated test before final deployment.

3.1.2.2.2. Upon successful review, the test will be deployed automatically and made available for immediate use.

Note: In autobuild, deployment occurs automatically. In manual deployment, developers handle the deployment process; users are not involved in deploying the test.

3.1.2.3. Notifications:

3.1.2.3.1. The system shall send notifications (via email and in-app) to users when the autobuild process is complete, indicating whether the test is ready for use or if any errors occurred.

3.1.2.3.2. Admins shall receive alerts when a new autobuild request is generated, as well as when manual intervention is required (e.g., failed builds).

3.1.3 Model Creation and Management

3.1.3.1. Request for New Models:

3.1.3.1.1. Users shall be able to submit requests for new models (e.g., new interview models or topics).

3.1.3.1.2. Like evaluation requests, the system shall validate that all required fields for model requests (e.g., model name, topic description) are completed before submission.

3.1.3.1.3. The system shall assign a unique identifier to each model request for tracking purposes.

3.1.3.1.2. Request Approval:

3.1.3.1.2.1. Admins shall be able to review all model requests and decide whether to approve or reject them.

3.1.3.1.2.2. Approved requests shall be assigned to the development team, who will manually build the requested model.

3.1.3.1.3. Autobuild for Models:

3.1.3.1.3.1. For supported model types, the system shall offer an autobuild option similar to evaluations, allowing the AI to generate and deploy the model without manual intervention.

3.1.3.1.4. Status Updates:

3.1.3.1.4.1. Users shall be able to track the status of their model requests (e.g., "Pending," "Built") through a dashboard.

3.1.3.1.4.2. Notifications shall be sent to users once the requested model is approved, built, or rejected, along with any relevant feedback.

3.1.4 Report Creation and Management

3.1.4.1. KPI Report Generation:

3.1.4.1.1. Users shall be able to create reports based on specific Key Performance Indicators (KPIs) by defining the required parameters (e.g., data source, time frame, target values).

3.1.4.1.2. The system shall validate that all necessary inputs for KPI creation (e.g., metrics, target goals) are filled out correctly.

3.1.4.2. Customization of Reports:

3.1.4.2.1. Users shall be able to customize the format of reports, including selecting the type of visualizations (charts, graphs)

3.1.4.2.2. Users shall have the option to export reports (PDF, Excel).

3.1.4.2.3. The system shall provide options to save report templates for future use.

3.1.4.3. Viewing and Exporting Reports:

3.1.4.3.1. Reports shall be viewable within the application, with interactive elements like drilldowns into detailed metrics.

3.1.4.3.2. Users shall have the ability to export reports in multiple formats (e.g., CSV, PDF, Excel) for external analysis or sharing.

3.1.4.4. Glossary and Description Field for Custom Objects:

3.1.4.4.1. The description field is essential in all custom objects, as it serves as input for prompt generation, guiding the LLM in generating code.

3.1.4.4.2. A glossary of key terms shall be provided to help users include relevant terms, improving the quality of custom objects, particularly for reports.

3.1.5 User Roles and Permissions

3.1.5.1. User Roles:

3.1.5.1.1. The system shall support multiple user roles (e.g., Admin, User, Viewer) with varying levels of access to features within the Studio module.

3.1.5.2. Admin Capabilities:

3.1.5.2.1. Admins shall have the ability to approve or reject requests for new evaluations and models.

3.1.5.2.2. Admins shall be able to view a list of all requests, filter by user or status, and generate summary reports of user activities.

3.1.5.3. User Access:

3.1.5.3.1. Users shall only be able to access their own requests and related statuses. Admins shall have full visibility into all user requests.

3.1.5.3.2. Role-based permissions shall restrict access to critical features such as approval of requests or access to KPIs to appropriate users.

3.1.6 Notifications and Alerts

3.1.6.1. Status Change Notifications:

3.1.6.1.1. The system shall send real-time notifications to users whenever the status of their requests changes (e.g., from "Pending" to "Built").

3.1.6.1.2. Users shall be notified via both email and in-app alerts for significant events such as request approval, build completion, or rejection.

3.1.6.2. Error Alerts:

3.1.6.2.1. If an error occurs during the autobuild process (e.g., failed test generation), the system shall notify both users and admins immediately, including a detailed error message.

3.1.6.3. Admin Alerts:

3.1.6.3.1. Admins shall receive alerts for any pending approval requests, rejected auto build processes, or failed manual builds.

3.2 Non-Functional Requirements

3.2.1. Performance:

3.2.1.1. The system shall handle at least x concurrent users without performance degradation.

3.2.1.2. Response time for any user action shall not exceed x seconds.

3.2.2. Usability:

3.2.2.1. The user interface shall be intuitive and user-friendly, with clear navigation.

3.2.2.2. Accessibility standards shall be met to ensure usability for all users.

3.2.3. Security:

3.2.3.1. The system shall implement secure user authentication and authorization mechanisms.

3.2.3.2. Data integrity and confidentiality must be maintained throughout all operations.

3.3 Business Rules

3.3.1. All requests for new evaluations and models must be tracked and logged within the system.

3.3.2. Users must have appropriate permissions to create evaluations and submit requests.

3.3.3. Evaluations and models must undergo a review process before they can be marked as "Built" or "Available."

3.3.4. KPIs generated through the reporting feature must adhere to organizational standards for metrics.

4. Assumptions & Constraints

4.1. Assumptions

4.1.1. Users will have the necessary training to utilize the Studio module effectively.

4.1.2. The underlying technology and infrastructure will support the implementation of AI features.

4.1.3. The development team will provide timely support for any technical issues during implementation.

4.2. Constraints

4.2.1. Any changes in user requirements after the initial implementation phase may lead to delays.

4.2.2. Resource availability, including skilled personnel and technology, may impact project timelines.

Investigative

Business Requirement Document (BRD)

Project Name	MagicAUDIT
Document Author	Chandrima Pal/Mahabub Hossain
Project Sponsor	[Sponsor's Name]
Stakeholders	[List of Key Stakeholders]
Version	[Version Number]

1. Executive Summary

1.1. Objective: Develop and implement an **Investigative system** to enable comprehensive evaluations of projects, products, and processes across industries.
1.2. Scope: The system will allow SMEs to assess **AI performance, reliability, and compliance** with established standards and benchmarks.
1.3. Purpose: SME feedback will provide **critical insights** to improve the AI's **effectiveness** and ensure **adherence to industry and organizational standards**.
1.4. Outcome: The evaluations conducted through the SME Audit system will contribute to the continuous **refinement and optimization** of AI systems for **better alignment with business goals**.

2. Business Objectives

The business objectives for the Investigative system include:

2.1. Provide an interface for assigning SME(s) to AI model evaluations in a clear and structured manner.
2.2. Allow users to select various parameters (topic, evaluation mode, domain) for more tailored evaluations.
2.3. To identify areas of improvement based on SME feedback.
2.4. Enable setting of priority levels for SME tasks to ensure timely completion of critical evaluations.
2.5. Provide real-time visibility into SME assignments, including progress, score, and feedback status.

2.6. To streamline the process of gathering, reviewing, and managing expert evaluations.
2.7. Enable SMEs to brainstorm issues and refine LLM prompts as a part of suggested improvements.

3. Requirements

3.1 Functional Requirements

3.1.1. The system must allow users to create, assign, and track evaluations for various domains for SMEs (e.g., technical, compliance, ethical, or performance-based audits).
3.1.2. It should support multiple evaluation modes such as Project specific and Core Model.
3.1.3. Users must be able to filter evaluations by domain, priority, or status, and track the progress of assigned audits.
3.1.4. The system must allow users to view SMEs feedback in a structured format, including scores and written comments.
3.1.5. SME Assignment Feature: Users must be able to create a new SME assignment by providing the following mandatory fields:
- **3.1.5.1.** Topic: The specific topic to be evaluated (e.g., "Bias").
- **3.1.5.2.** Evaluation Mode: Whether the evaluation is "Project-Specific" or more generalized.
- **3.1.5.3.** Domain Tag(s): Categories or industries the audit is related to (e.g., "Insurance").
- **3.1.5.4.** Evaluator Model: The specific model or system being evaluated, which SMEs can search for.
- **3.1.5.5.** No. of Questions: The number of questions in the evaluation (e.g., "5").
- **3.1.5.6.** SME Assignment: The SME's contact information (e.g., email) to whom the evaluation will be assigned.
- **3.1.5.7.** Priority: The level of importance for the assignment (e.g., "High").
- **3.1.5.8.** Requested ETA: The expected completion date for the audit assignment.

3.1.6. Assignments List Visibility: After creating an assignment, it will automatically be displayed in the assignments list, with details such as:
- **3.1.6.1.** SME's name or contact information.

3.1.6.2. Assigned Topic.
3.1.6.3. Evaluation Mode (e.g., Project-Specific, Core Model).
3.1.6.4. Domain Tag(s) (If selected Project-Specific).
3.1.6.5. Priority.
3.1.6.6. Number of Questions.
3.1.6.7. Assigned Date.
3.1.6.8. Requested ETA.
3.1.6.9. Status (e.g., In Progress, Done, New).

3.1.7. Audits must have assignable priorities and due dates, allowing users to prioritize based on importance.

3.1.8. Users should be able to export audit results for further analysis and reporting.

3.1.9. Future Implementation

The current SME functionality is basic, and future directions may include the following enhancements:

3.1.9.1. Integration of vendor organizations for outsourcing audits.

3.1.9.2. Implementation of layered approvals within well-defined organization-specific workflows.

3.1.9.3. The ability for certain domains, such as healthcare, to configure additional scores for SMEs to rate on, beyond just the overall score.

3.2 Non-Functional Requirements

3.2.1. Usability: The platform must be intuitive and require minimal training for SMEs to perform evaluations efficiently.
3.2.2. Performance: The system should provide fast and responsive feedback to user inputs, even under high concurrent use.
3.2.3. Security: It must ensure that sensitive information is secure and complies with industry standards, such as data protection regulations.
3.2.4. Scalability: The system should be capable of handling increasing numbers of users, evaluations, and feedback data without performance issues.

3.2.5. Reliability: The platform must ensure that all feedback is stored reliably and can be retrieved when needed, with regular backups in place.
3.2.6. Accessibility: The system must be accessible on various devices (desktop, tablet, mobile) and be compliant with accessibility standards (e.g., WCAG).

3.3. Business Rules

3.3.1. Field Identification and Requirements for Each Project

3.3.1.1. Topic (Mandatory):
3.3.1.1.1. Description: Specifies the exact topic being evaluated by the SME. Currently available topics are-
3.3.1.1.1.1. Bias
3.3.1.1.1.2. Safety
3.3.1.1.1.3. Ethics
3.3.1.1.1.4. Explainability
3.3.1.1.1.5. Hallucination
3.3.1.1.2. UI Validation:
3.3.1.1.2.1. Cannot be left empty.
3.3.1.1.2.2. Must be selected from predefined dropdown options
3.3.1.1.2.3. Maximum one topic per assignment.
3.3.1.2. Evaluation Mode (Mandatory):
3.3.1.2.1. Description: Defines whether the audit is-
3.3.1.2.1.1. Project-Specific.
3.3.1.2.1.2. Core Model.
3.3.1.2.2. UI Validation:
3.3.1.2.2.1. Must be selected from predefined dropdown options (Project-Specific, Core Model).
3.3.1.2.2.2. Default value: None (user must select a value).
3.3.1.3. Domain Tag(s) (Mandatory, not applicable for Core Model):
3.3.1.3.1. Description: Categories or industries the project audit is related to, Current available tags are-
3.3.1.3.1.1. Insurance
3.3.1.3.1.2. Finance
3.3.1.3.1.3. Information Technology

3.3.1.3.2. UI Validation:

3.3.1.3.2.1. Must allow for multiple tag selection.

3.3.1.3.2.2. Only valid tags (predefined in the system) can be used.

3.3.1.3.2.3. Users must be able to select from a dropdown.

3.3.1.3.2.4. When selected Core Model this field is getting disabled, restricting user to provide any input.

3.3.1.4. Evaluator Model (Mandatory):

3.3.1.4.1. Description: Specifies the AI model being evaluated by the SME.

3.3.1.4.2. UI Validation:

3.3.1.4.2.1. Cannot be left empty.

3.3.1.4.2.2. Must be searchable, with suggestions displayed as the user types.

3.3.1.4.2.3. Must allow for selecting predefined models only.

3.3.1.5. Number of Questions (Mandatory):

3.3.1.5.1. Description: The number of questions for the evaluation (e.g., "5").

3.3.1.5.2. UI Validation:

3.3.1.5.2.1. Must be a positive integer.

3.3.1.5.2.2. Cannot be zero or negative.

3.3.1.5.2.3. Maximum allowed value: 100.

3.3.1.6. SME Assignment (Mandatory):

3.3.1.6.1. Description: The SME's contact information (e.g., email) to whom the evaluation will be assigned.

3.3.1.6.2. UI Validation:

3.3.1.6.2.1. Must be a valid email address format (e.g., name@example.com).

3.3.1.6.2.2. Cannot be left empty.

3.3.1.6.2.3. Must be selected from predefined dropdown options ensure that the same SME isn't assigned multiple times for the same evaluation and only selected SME is able to evaluate his assignment.

3.3.1.7. Priority (Mandatory):

3.3.1.7.1. Description: Defines the importance of the audit -

3.3.1.7.1.1. High

3.3.1.7.1.2. Medium

3.3.1.7.1.3. Low

3.3.1.7.2. UI Validation:

3.3.1.7.2.1. Must be selected from predefined dropdown options.

3.3.1.7.2.2. One option is selectable.

3.3.1.8. Requested ETA (Mandatory):

3.3.1.8.1. Description: Expected completion date for the audit assignment.

3.3.1.8.2. UI Validation:

3.3.1.8.2.1. Must be a valid date format, selected from the Calander (e.g. `DD/MMM/YYYY`).

3.3.1.8.2.2. Cannot be in the past.

3.3.2 UI Validation Rules Overview

3.3.2.1. Mandatory Fields: Fields like **Topic**, **Evaluation Mode**, **Evaluator Model**, **Number of Questions**, **SME Assignment**, **Priority**, and **Requested ETA** must be filled in before submission. The system should display error messages if any mandatory fields are left blank or contain invalid input.

3.3.2.2. Optional Fields: Fields like **Domain Tags** can be left blank if selected Core Model.

3.3.2.3. Data Types: Ensure fields like **Number of Questions** are restricted to integer inputs, while **Priority, Evaluation Mode, Topic, Domain Tag(s) & SME Assignment** are selected from dropdowns with valid options.

3.3.2.4. Date Validation: The **Requested ETA** must be a future date, and user should be restricted if an invalid or past date is selected.

3.3.3 Audit Submission Rules

3.3.3.1. The audit can only be submitted when all **mandatory fields** are populated with valid data.

3.3.3.2. Users attempting to submit incomplete forms must receive clear error messages guiding them to correct specific issues.

3.3.3.3. Once submitted, only **authorized personnel** can edit the audit details, and changes must be logged for traceability.

3.3.3.4. SME access will be role-based, ensuring that only authorized users can access certain evaluations or domains.
3.3.3.5. Each audit must have a predefined timeframe for completion, after which it should be flagged for follow-up if not completed.

4. Assumptions & Constraints

4.1. Assumptions

4.1.1. SMEs will be provided with sufficient information and resources to complete evaluations accurately.
4.1.2. The system will be integrated into existing workflows, allowing SMEs to conduct audits without disruptions.
4.1.3. SMEs will have the necessary expertise and knowledge to evaluate the relevant topics.
4.1.4. The audit system will be flexible enough to accommodate different industries or audit types (e.g., technical, compliance, or quality assurance audits).

4.2. Constraints

4.2.1. The system must comply with any applicable regulatory or compliance requirements, especially when used in industries with strict guidelines (e.g., healthcare, finance).
4.2.2. There may be limitations in integrating the system with other platforms or tools already in use within the organization.
4.2.3. System performance may be impacted if the number of concurrent users or audits exceeds projected limits.
4.2.4. The platform must operate within the organization's IT infrastructure capabilities, including server capacity and bandwidth limitations.

Business Requirement Document (BRD)

Project Name: AI Governance Metrics Dashboard

Document Version: 1.0

Date: November 2025

Prepared by: Mahabub Hossain

1. Purpose

The purpose of this document is to define, structure, and standardize AI performance and governance metrics to ensure safety, transparency, and accountability in AI systems. The KPIs will form part of the MagicAudit platform, supporting both reflective and investigative testing across projects.

2. Objectives

• Establish measurable indicators for AI technical safety, operational risk, compliance, and ethical impact.

• Enable monitoring of AI models and use cases over time.

• Integrate these KPIs into dashboards and reports to support AI maturity assessment and regulatory alignment.

3. Scope

The scope includes defining metrics, their calculation logic, and data sources across four key domains: Technical Safety, Operational Risk, Compliance & Governance, and Societal & Ethical Metrics.

4. Metric Categories and Definitions

4.1 Technical Safety Metrics

Metric	Description	Calculation / Source	Notes
Efficacy vs Safety Tradeoff	Measures balance between model accuracy and safety (e.g., average daily efficacy vs. safety scores).	Objective, Reflective, Investigative testing average score results comparing efficacy and safety metrics.	Visualized as trend or scatter graph.
Hallucination Rate	Frequency of AI-generated	Topic: Hallucination; ratio	Track weekly/monthly.

	inaccurate or fabricated outputs.	of flagged incidents over total responses. | Track the hallucination scores & topics and show average over a period.	
Toxicity / Harmful Output Rate	Frequency of unsafe or offensive outputs.	Average toxicity score by topic or model; visualized as time-series.	Example: Average score and counts per week.
Explainability Score (Future)	% of predictions with interpretable explanations (e.g., SHAP, LIME).	Derived from explainability module; average across test samples.	Aggregated under 'Explanations available'.
Drift Detection (Data & Concept Drift) (Future)	Detects deviation in data distribution or concept boundaries over time.	Evaluate changes in data variance or config thresholds.	Captures both data drift and concept drift.
Fail-safe Trigger Rate (Future)	Frequency of emergency stop or fallback mechanisms triggered.	Prescriptive AI triggers (future feature).	Currently N/A; placeholder for future system alerts.

4.2 Operational Risk Metrics

• # of AI Incidents per Month – safety, efficacy, maturity- threshold breach. Monthly count and classification of incidents. (Similar to incident trend line)

• Mean Time to Detect (MTTD) – Average time between occurrence and identification. - depends on Prescriptive. (Future)

• Mean Time to Mitigate (MTTM) – Average time to resolve identified issues.- depends on Prescriptive. (Future)

• % of AI Projects with Human-in-the-Loop Oversight – While creating a project, add an option for human in loop, if selected, add high value in the maturity. (Future implementation- many of the use cases AI will not solve the problem 100% but will utilize AI or agents in such a way that there will be a workflow where a human will also interact. Applicable for complex projects. project type with human in loop, the

project will get a very high level of maturity. a project has human in loop and then we can use it into the maturity calculation.
)

• % of Projects with Red-Team Testing – Projects tested through investigative Evaluation.

4.3 Compliance & Governance Metrics

• % Projects with Completed Documentation – Projects with model cards or datasheets.

• Audit Pass Rate on Fairness/Bias Checks – total bias eval vs % of bias evaluations crossing threshold. (Similar to incedent trendline bias), ~~need to show no of total project vs no of projects where bias threshold breaches.~~

• % AI Systems Mapped to Regulatory Obligations – Mapped to frameworks (EU AI Act, NIST RMF, etc.).- NO of projects that executed compliance related evaluation vs no of total onboarded projects.

• Model Inventory Completeness Ratio – Models tracked within governance framework.

4.4 Societal & Ethical Metrics

• Bias Index – No of evaluation that croses bias threshold vs no of bias evaluation. (Strictly based on Topic: Bias)

• Equity Impact Score – Degree of positive/negative impact on ownership and inclusion.

• Environmental Footprint – Compute cost, energy usage, or emissions per training run.

• Public Sentiment / Trust Score – Derived from surveys or stakeholder perception.

5. Data & Logic Summary

Primary Source: Reflective & Investigative Test Results table.
Frequency: Monthly aggregation.
Score Range: 0–10 normalized.
Processing: Group results by month and client, filter by topic, compute averages, and store for reporting.

6. Reporting and Visualization

Report	Description	Visualization Type
Bias & Fairness Trend	Monthly average bias scores across clients.	Line Chart
Explainability Coverage Gauge	% of projects with explainability implemented.	Radial Gauge
Compliance Heatmap	Project readiness against compliance frameworks.	Heatmap
Operational Incidents Dashboard	Tracks MTTD, MTTM, and safety incidents.	Bar/Line Chart

7. Dependencies

- Integration with Reflective & Investigative Testing modules.
- Access to Model Registry and Project Assessment data.
- Configuration file for concept boundaries (for drift metrics).
- Optional linkage to sentiment survey and compute metadata.

8. Future Enhancements

- Enable automatic fail-safe triggers and alerts.
- Develop dedicated Equity & Environmental evaluation modules.
- Introduce longitudinal trend reports for bias and explainability evolution.

Reference: Discussion on KPIs.docx

Business Requirement Document (BRD)

Efficacy vs Safety Tradeoff

This KPI **compares two dimensions of AI performance** for each model or use case:

Dimension	Meaning	Example Metric Type
Efficacy (Accuracy / Usefulness)	How well the model performs its intended task — e.g., how accurate, coherent, or relevant its outputs are.	All Efficacy Scores
Safety (Robustness / Risk)	How *safe* or *trustworthy* those outputs are — i.e., do they avoid harmful, toxic, or sensitive content.	All Safety Scores

⚖️ What the Tradeoff Means

In many real-world AI use cases, there's a **tension between being highly effective and being completely safe**:

- A model that's **very effective** (high accuracy, creative, expressive) may also **take more risks**, occasionally producing unsafe outputs.
- A model that's **very safe** (strict filters, conservative answers) might produce **less useful** or less detailed responses.

This KPI seeks to **quantify that balance.**

📊 How It's Used

For each model or use case:

1. Compute **Efficacy Score.**
2. Compute **Safety Score.**
3. Plot both or calculate a ratio/correlation between them.
    a. Example: showing models distributed across high/low efficacy vs high/low safety.

This helps identify:

- Cases where **efficacy is high but safety is low** → ⚠️ risky models.
- Cases where both are high → ✅ ideal balance.
- Cases where both are low → 🔧 poor-performing models needing improvement.

🧩 Purpose in KPI Dashboard

The KPI **"Efficacy vs Safety Tradeoff"** therefore answers the question:

"How well does each AI system maintain usefulness without compromising safety?"

It helps stakeholders:

- Detect **imbalances** (high performance but unsafe outputs).
- Prioritize **risk mitigation efforts** where needed.
- Showcase **responsible AI design** by demonstrating controlled balance.

Updated Explanation

This metric captures the balance between how effective a model is versus how safe its outputs are. Efficacy (or accuracy) refers to how well the model performs its intended task — for example, the accuracy, usefulness, or coherence of a summary. Safety (or robustness) measures how risk-free or trustworthy those outputs are — for example, whether they avoid harmful, biased, or sensitive content.

In practice, there are often situations where a model shows very high efficacy but also produces unsafe or non-compliant outputs. This KPI helps visualize and measure such tradeoffs by comparing or correlating efficacy scores with safety scores across models or use cases.

The goal is to find an optimal balance: models that achieve high efficacy while maintaining strong safety standards. It can be represented as a scatter or trend plot showing where each model or use case lies on the efficacy–safety spectrum.

Business Requirements Document (BRD)

Feature Name: Governance Analytics — Bias & Fairness Trend, Explainability Coverage, Compliance Heatmap

Application: magicAudit

Version: 1.0

Document Author: Mahabub

Date: 28-Oct-2025

1. Overview

This enhancement introduces three analytical modules under the Governance Analytics suite of magicAudit. These modules provide governance teams with insights into AI fairness, explainability adoption, and compliance readiness across onboarded projects and clients.

Functionalities Covered:
1. Bias & Fairness Trend — Average bias score over the last 6 months.
2. Explainability Coverage Gauge — Percentage of AI projects with explainability tests executed.
3. Compliance Heatmap — Project-wise readiness visualization across major compliance frameworks (e.g., NADA, EU AI Act, NIST RMF, GDPR).

2. Purpose

The objective is to provide a unified governance dashboard where decision-makers can monitor the fairness, interpretability, and compliance alignment of AI systems.

Specifically:
- Track fairness stability trends over time.
- Measure adoption and execution frequency of explainability assessments.
- Identify gaps in compliance coverage across frameworks and projects.

3. Scope

In Scope:

- Data aggregation from Reflective & Investigative Testing categories.
- 6-month trend analysis for bias and fairness.
- Explainability execution tracking by project.
- Compliance readiness scoring per project and framework.
- Visualization components (bar, pie, heatmap).

Out of Scope:

- Real-time monitoring (dashboards update on scheduled intervals).
- Model-level segregation within projects.
- Manual compliance document validation.

4. Feature 1 — Bias & Fairness Trend (6-Month View)

Shows the average monthly score of "Bias" topic (from Reflective & Investigative Testing results) for the last six months.

Data & Logic:
- Source: Reflective & Investigative Test Results Table
- Score Range: 0–10
- Calculation:
  - Group test results by Month and Client
  - Filter only "Bias" topic
  - Compute average bias score for each month
  - Display 6-month trend (latest month = current month)

Visualization: Bar Chart (Month vs Average Bias Score)

5. Feature 2 — Explainability Coverage Gauge

Represents the coverage percentage of projects that have executed Explainability-related tests at least once compared to all onboarded projects.

Data & Logic:
- Source: Reflective & Investigative Test Results
- Metrics:
  - Explainability Run Count = Number of times "Explainability" topic executed
  - Total Run Count = All tests executed
  - Projects with Explainability = Projects where Explainability topic has ≥ 1 run
- Calculation: Coverage % = (Projects with Explainability / Total Projects) × 100

Drilldown Details:
Shows project-level breakdown with total vs explainability runs.

Visualization:
- Gauge / Pie Chart — % of Projects with Explainability
- Bar Chart Drilldown — Project-wise explainability run ratio

6. Feature 3 — Compliance Heatmap

Displays project-wise compliance readiness against selected frameworks such as NADA, EU AI Act, NIST RMF, GDPR, etc.

Data & Logic:
- Source: Reflective & Investigative Test Results
- Each framework is associated with a set of topics (e.g. Car Dealership).
- For each project:

- Identify topics mapped to each framework.
- Calculate average score (0–10) for those topics.
- Populate heatmap cell with that score.

Visualization:
- Type: Heatmap
- X-Axis: Frameworks (NADA, EU AI Act, NIST RMF, GDPR)
- Y-Axis: Project Names
- Cell Value: Average Score (color-coded gradient: red = low, green = high)

Business Requirements Document (BRD)

Feature Name: Total vs High Severity Incidents — AI Testing Metrics Overview

Application: magicAudit

Version: 1.0

Document Author: Mahabub

Date: 27-Oct-2025

1. Overview

The 'Total vs High Severity Incidents' dashboard in magicAudit provides a client-level view of testing outcomes across all onboarded AI projects. It highlights the number of total tests executed versus those that fall below a configurable severity threshold, helping governance teams monitor testing stability and identify critical performance issues across clients.

2. Purpose

This feature enables audit and governance teams to:
- Identify clients with frequent high-severity testing incidents.
- Track the overall testing quality and incident rates.
- Assess which test categories (Objective, Reflective, Investigative) are most prone to severe issues.

3. Scope

In Scope:

- Aggregation of all test results across all projects within a client.
- Identification and tracking of high-severity test results (below configured threshold).
- Graphical display of total vs high-severity incidents for the last six months.
- Filtering of metrics by test category or topic (Objective Scores, Reflective Topics, Investigative Topics).

Out of Scope:

- Individual test case details or execution logs.
- Project-Model-level filtering.
- Real-time alerts (to be covered in future enhancements).

4. Definitions

Term	Description
High Severity Incident	Any test score below the configured threshold (default = 5).

Client-Level Aggregation	Consolidation of test results across all projects under a single client.

5. Business Logic

Step 1 — Data Aggregation

Collect all test results (Objective, Reflective, Investigative) from all projects under a single client for the selected reporting period (last 6 months).

Step 2 — Severity Threshold Check

For each test result, check if the score is below the configured threshold (default = 5). If yes, count it as a High Severity Incident. Threshold is configurable via the Admin Module.

Step 3 — Summary Metrics

- Total Tests Executed = Count of all tests in selected period.
- High Severity Incidents = Count of tests with score < threshold.
- % High Severity = (High Severity / Total Tests) × 100.

Step 4 — Visualization

- Monthly trend for last 6 months showing total vs high-severity incidents.
- Filter to switch between Objective, Reflective, or Investigative test categories.
- Option to drill down to specific score or topic metrics.

6. Configuration Parameters

Parameter	Description	Default	Module
High Severity Threshold	Minimum acceptable score for tests (below this = high severity)	5	Admin Module
Reporting Window	Time duration for trend visualization	6 months	System Setting

7. Output Metrics & Visualization

Dashboard Components:
- Line/Bar Chart: Monthly total vs high-severity incidents (6 months).
- Filter by Category: Objective / Reflective / Investigative.
- Drill-down view: Metrics by specific score (Objective) or topic (Reflective & Investigative).
- KPI Indicators: Total Tests, High Severity Incidents, % High Severity.

Export Options: CSV / Excel / PDF

9. User Roles & Permissions

Role	Access	Description
Admin	Configure severity threshold and manage client settings.	Can modify threshold value and reporting window.
Governance	View client-level incident metrics and export reports. Analyze incident trends and investigate root causes.	Read & Write access. Access to drill-down metrics and trend visualizations.

10. Success Criteria

- Accurate aggregation of all project test results under each client.
- Correct identification of high-severity incidents based on threshold.
- Interactive visualization of total vs high-severity trends for 6 months.
- Configurable parameters functioning as expected.

11. Future Enhancements (Next Phase)

- Add anomaly detection to automatically flag spikes in high-severity incidents.
- Integrate real-time alert system for severe test failures.

Business Requirements Document (BRD)

Feature Name: Governance Maturity Overview — AI Projects by Test Coverage

Application: magicAudit

Version: 2.0

Document Author: Mahabub

Date: 27-Oct-2025

1. Overview

The Governance Maturity Overview — AI Projects by Test Coverage module in magicAudit provides a summarized view of all onboarded AI projects, categorized by their testing maturity. This feature measures how comprehensively each project has undergone three core testing categories — Objective, Reflective, and Investigative Testing — and determines whether a project is Well Tested or Under Tested based on configurable thresholds.

2. Purpose

This feature enables governance and compliance teams to:
- Track the testing health of all AI projects.
- Identify projects that lack sufficient testing coverage.
- Support audit and maturity reporting within AI Governance dashboards.

3. Scope

In Scope:

- Evaluation of AI project test coverage at the project level (not individual models).
- Classification of each project as Well Tested or Under Tested based on defined rules.
- Configurable test frequency thresholds managed via the Admin Module.
- Dashboard/summary visualization of test distribution across projects.

Out of Scope:

- Model-level testing metrics (e.g., Google AI vs. OpenAI within a single project).

4. Definitions

Term	Description
Well Tested Project	A project that meets all success and frequency criteria for all test categories.
Under Tested Project	A project that fails to meet one or more

	criteria.

5. Business Logic

Step 1 — Category Check

For each onboarded project, check if each testing category (Objective, Reflective, Investigative) has been executed successfully at least once. If any category has never been successfully executed → mark project as Under Tested.

Step 2 — Frequency Check

If all three categories have been run successfully at least once, check if each category has been tested successfully ≥ [Configurable Minimum Threshold] times (default = 15). This threshold is configurable via the Admin Module.

Step 3 — Classification

- If both Step 1 and Step 2 are TRUE → Project = Well Tested
- Otherwise → Project = Under Tested

6. Configuration Parameters

Parameter	Description	Default	Module
Min Test Frequency per Category	Minimum number of successful runs required for each testing category	15	Admin Module
Project Evaluation Scope	Determines if the check is at project or model level	Project	System Constant

7. Data Sources

Data Entity	Description
Project Table	Contains all onboarded AI projects and their metadata.
Test Results Table	Stores all test run results with category, status.
Configuration Table	Contains admin-configured thresholds and settings.

8. Output Metrics & Visualization

Dashboard Widgets:

- Pie Chart: Distribution of Well Tested vs Under Tested projects.
- Bar Graph: Test frequency per category across all projects.
- Filter Options:
  - Date Range
  - Test Category

Export Options: CSV / Excel / PDF

9. User Roles & Permissions

Role	Access	Description
Admin	Configure thresholds, view all projects.	Can modify test frequency threshold.
Governance	View project test coverage dashboard, Run Evaluation.	Read & Write access.

10. Success Criteria

- Accurate categorization of each project into Well Tested / Under Tested.
- Configurable parameters working as intended.
- Real-time reflection of new test results on dashboard metrics.

11. Future Enhancements (Next Phase)

- Extend coverage evaluation to model-level testing maturity.
- Automated alerts for projects trending toward under-tested status.

Business Requirement Document (BRD)

Project Name: MagicAudit - AutoDemo Onboarding

Document Author: Mahabub Hossain

Version: [2.0]

1. Executive Summary

The AutoDemo Onboarding process enables clients to request and experience an interactive demo, followed by automatic trial setup. It aims to improve demo conversion, simplify onboarding, and automate the Demo -> Trial -> Subscription process.

2. Business Objectives

- Increase Client Conversion through self-driven demos.
- Enhance Customer Experience via personalized, interactive bot demos.
- Standardize & Automate Onboarding workflows.
- Optimize Resource Utilization with automated outboarding.

3. Functional Requirements

3.1 Demo Flow

- Instant Demo Request via mA Site.
- Automated Demo Link & Verification (Email/SMS).
- Demo Form captures client details and preferences.
- Interactive DemoDJ Page executes bot-guided demo using DemoDJ Class.
- Post-demo, trial credentials emailed and Trial ClientDB created via OnBoarder.

3.2 Onboarding & Transition

- Trial to Subscription conversion with billing integration.
- Role-based Access control.
- Automated Outboarding and data cleanup post-trial.

3.3 Backend Components

Component	Description	Expected Function
mA Site	MagicAudit website entry point for demo requests	Accept instant demo requests, validate, and trigger AutoDemo flow
Demo Link Email/SMS	Automated communication sent post-request	Send demo link and verification code

Demo Form	Web form for collecting user and demo preferences	Capture client info for tailoring demo experience
DemoDJ Page	Core demo interface with interactive bot	Execute scripted demo with user interaction
OnBoarder Class/Agent	Backend module to create client-specific databases	Build demo/trial databases from template
DemoDJ Class/Agent	Executes demo scripts and interacts with users	Orchestrate live demo guided by automation bot
Template DB	Base database with pre-configured demo data	Used as reference for generating client DBs
Demo ClientDB	Temporary DB for demo execution	Mirrors client context during demo
Trial ClientDB	Auto-generated DB for client trial	Enables continued engagement post-demo
Trial Link Email	Follow-up email post-demo	Sends trial access credentials and details

4. Non-Functional Requirements

- Reliability: 99.9% uptime.
- Scalability: Handle concurrent demos.
- Security: Encryption and verification.
- Usability: Quick, intuitive, guided flow.

5. Business Rules

- Trial Period: 7-15 days.
- Account Conversion: Requires valid payment info.
- Outboarding: Cleanup 3 days post-trial.
- Verification: Mandatory before demo.
- Role Assignment: Auto-applied based on trial type.

6. Assumptions & Constraints

Assumptions:

- Clients understand product basics.
- Flow is fully automated.
- Clients have valid contact info.

Constraints:

- Must fit current architecture.
- Subscription updates must not break onboarding.
- Limited manual support available.

7. Expected Outcomes

- Reduced manual effort.
- Improved demo-to-trial conversions.
- Enhanced client engagement.
- Optimized system resources.

https://pursuitsoftwarebiz-my.sharepoint.com/:p:/g/personal/pravin_d_pursuitsoftware_biz/EUINFbgiyWlKn77E_iFIMG8B00Q39_KhdylaW6Aj0y0bQg?e=xnyyPg

1. Overview

The Admin Module is designed to empower clients with full control over their platform configuration, access, and operational governance. It is composed of three major sections: **Security**, **General Settings**, and **Specific Settings**, each responsible for distinct aspects of administrative functionality.

- **Security Module**: Handles access control, user and token manag
- **General Settings**: Enables JSON-driven configuration of global platform behavior, including model-feature-score relationships and maturity logic.
- **Specific Settings**: Offers granular control over testing strategies, analytics, and module-specific behaviors.

This BRD details each of these sections individually to maintain clarity while capturing a complete scope.

2. Security Module

2.1 Overview

The Security module within the Admin Panel is responsible for managing core access, authentication, and auditing capabilities of the platform. It ensures that clients can maintain secure operations by controlling user access, API token management, and monitoring usage and activities.

Sub-modules:

- User Management
- Access Management
- API Token Management
- Logs and Auditing

2.2 Sub-Modules & Features

2.2.1 User Management

Objective: Allow clients to manage their own users with basic CRUD (Create, Read, Update, Delete) operations. (Only Admin can perform these actions)

Features:

- **Add User**: Form to create a new user including username, email, role (Admin, Governance, SME, Studio), password, and optional profile picture.
- **Edit User**: Modify user details such as role and email.
- **Delete User**: Soft delete preferred, or hard delete.
- **List Users**: Table/grid view of all users with sorting, filters, and actions.

Example Use Case: A client admin can add team members, assign roles, and manage their access over time.

2.2.2 Access Management

Objective: Define and manage what modules or features a user can access.

Features:

- Assign access rights per user for Admin, Governance, SME, Studio modules.

Example Use Case: A user from Client X can be restricted to SME, while another accesses Studio and Governance.

2.2.3 API Token Management

Objective: Provide secure API token generation and management.

Features:

- **Generate Token**: With expiry, scopes, and user/client mapping.
- **Revoke Token**: Invalidate tokens.
- **View Active Tokens**: Includes issue date, expiry, and usage metadata.
- **Future Enhancements**: Token algorithm selection, expiry defaults, limit on concurrent tokens.

Example Use Case: A backend service uses a token to pull results from APIs.

2.2.4 Logs and Auditing

Objective: Maintain a record of user and system activities.

Features:

- **Activity Logs**: Track actions with timestamps and user info.
- **Login History**: Include IP, device, browser.
- **Error Logs**: With severity and message.
- **Filters and Export**: By user, type, date, exportable to CSV/JSON.

Example Use Case: An admin reviews logs to detect unauthorized actions.

2.3 UI/UX Considerations

- **Role-Based Views**: Restricted to user permissions.
- **Client Isolation**: Strict tenant-based boundaries.

2.4 Technical Considerations

- **JSON-Driven Configuration (A1 Object)**
- **Frontend + Backend Logging**
- **Permission Checks for Sensitive Actions**

3. General Settings Module

3.1 Overview

This module governs platform-wide configurations editable by clients through the frontend UI. All changes impact the system globally and emphasize "No Dev Intervention Required."

3.2 Features

3.2.1 Model, Feature & Score Relationship Configuration

Clients will configure the underlying relationships between models, features, and scores via a UI linked to a JSON configuration file.

Functions:

- Add/Edit/Delete models (e.g., DeepSeek, Scientific Reasoning)
- Associate features (e.g., Summarization)
- Map scores (e.g., Fluency, Coherence)
- Real-time updates to JSON

Use Case: Modify Summarization to associate Fluency instead of Utility.

Why It Matters:

- Client-driven evolution
- Developer-free scaling
- Rapid onboarding of new AI models

3.2.2 Maturity Configuration

Purpose: Allow clients to define maturity assessment logic via weighted dimensions like Agentic, RAG, Prompt Evaluation.

Functionality:

- Assign weights to maturity types
- Fully customizable per client

Outcome: Tailored evaluations per client's business focus.

3.2.3 API Key Management for Models

Purpose: Manage API keys and access credentials per model.

Functionality:

- Add/Edit/Delete keys

- Link to models
- Set expirations and access rules

Enhancement Question: How to notify when key quota is exhausted?

3.2.4 Global Settings Philosophy

All settings here:

- Affect global behavior
- Are Admin-only
- Are synchronized via the A1 JSON object

3.3 Summary

- Client-centric
- UI-powered config updates
- Flexible and scalable

4. Specific Settings Module

4.1 Overview

Specific Settings allow granular configuration of platform behavior specific to project types and testing methodologies.

Categories:

- Project Settings
- Objective Testing
- Reflective Testing
- Investigative Testing
- Analytics & Prescription
- Studio
- SME

4.2 Details per Category

Project Settings

- Define metadata
- Set scoring inheritance

Objective Testing

- Select scores
- Pull/edit related config from JSON
- Example: Adjust "Text Quality" thresholds

Reflective Testing

- Similar to Objective
- Example: Tweak bias score logic or Q/A similarity thresholds

Investigative Testing

- Same behavior as above

Analytics & Prescription

- Control insights logic
- Define weights and thresholds

Studio

- Workflow for developer requests
- Toggle UX/UI settings

SME

- Evaluation and review configurations
- Includes tagging and scoring rule settings

4.3 UI/UX Behavior

- Dynamic load of JSON-driven forms
- Admins can edit, preview, and save thresholds or logic

4.4 Summary

- Modular, score-based control
- JSON-backed logic enforcement
- Clean UI for precise governance

cBusiness Requirement Document (BRD)

Module: Objective Testing

Categories: PII, PHI, PCI, Absence of Bad Words

1. Objective

Enable administrators to manage and configure threshold-based scoring logic for objective compliance categories (PII, PHI, PCI) and offensive language (Bad Words). This is to allow transparent evaluation and flexible updates to each category's parameters.

2. Stakeholders

- Product Owner
- Admin Users
- Development Team
- QA Team
- Compliance & Security Team

3. Functional Requirements

3.1 Score Selection

- A dropdown will be available for the admin to select from:
    - **No PII**
    - **No PHI**
    - **No PCI**
    - **Absence of Bad Words**

- **WEC**
- **WLL**
- **No Audio Pause**

3.2 Configuration Display & Editability

For PII / PHI / PCI *weightage.json*

- Upon score selection, all existing parameters (from JSON) will be listed in a tabular form:
  - **Parameter Name**
  - **Associated Weight (numeric value)**
- Admin can:
  - Edit weight values
  - Add new parameters
  - Remove existing parameters

For Absence of Bad Words

- System will pull from a plain-text list of bad words (profanity_wordlist 1.txt).
- The UI will:
  - Show each bad word as a list item
  - Allow admins to:
    - Add new bad words
    - Remove existing words from the list
- Any word present in this list is treated as **prohibited**, and content containing it will affect scoring.

WLL & WEC (Audio Silence Detection)

These settings are used to detect pauses in audio for various listening-based tasks.

Parameter	Type	Notes
minsilencelen	Number	Minimum silence length
silencethresh	Number	Can be negative

buffer	Decimal	Max value: **1**

No Audio Pause

Dedicated settings for detecting absence of significant pauses in speech.

Parameter	Type	Notes
minsilencelenNopause	Number	
silencethreshNopause	Number	Can be negative

Reflective Testing

This configuration applies globally to all reflective topics.

Parameter	Type	Notes
thresholdSimilarity	Decimal	Max value: **1.0**

Investigative Testing

This configuration applies globally to all investigative topics.

Parameter	Type	Notes
SummaryLength	Number	Currently set to **50**

3.3 Save Functionality

- Once edited:
  - Updated configurations are saved back into the appropriate JSON or text structure.
  - The system validates all entries:

    - For thresholds (PII/PHI/PCI): numeric values are required.
    - For bad words: only alphabetic or valid special characters (no blank entries).
- The data is persisted for backend processing and audits.

4. Data Sources

PII (No PII Score) – Example:

Parameter	Weight
Email	2
Contact Number	2
Social Security Numbers	3

PHI (No PHI Score) – Example:

Parameter	Weight
Disease	2
Test Results	2
Doctor's Name	1

PCI (No PCI Score) – Example:

Parameter	Weight
Credit Card Number	2
OTP	3
CIBIL Score	1

Bad Words List (Absence of Bad Words Score) – Example:

```
anal
asshole
bitch
cumshot
fuck
... (and more from the list)
```

5. Non-Functional Requirements

- Interface must support bulk additions (e.g., upload or multiline input for bad words).
- Audit logs must track who modified what and when.
- Real-time validation and confirmation messages post-save.
- Performance must remain optimal even with 1000+ bad words.

6. Future Enhancements

- Group bad words by category or severity.
- Support regex patterns in bad word matching.
- Enable scoring strategy preview with sample inputs.

AI Explainability Report

1. Rationale for AI Explainability and Need for the New Method

AI explainability remains a critical challenge because users typically receive answers from models without insight into how these answers were derived. Users can only assess correctness superficially, leading to uncertainty and mistrust, especially when unexpected or incorrect outputs occur. The existing method—comparing keyword overlaps between the outputs of target and reference models—is insufficient because it inadequately captures nuanced reasoning, especially when correct answers are phrased differently by different models, resulting in misleadingly scores. Hence, we introduce a new mechanism, attribute graph, explicitly leveraging the reasoning processes behind model outputs. By understanding the reasoning steps explicitly, users gain transparency, easier understanding of model's behavior and why model deliver correct or wrong answer as the final output.

2. Description and Functionality of Attribute Graph

The proposed method centers on generating structured attribute graphs based on reasoning steps used for generating model outputs. Each graph consists of:

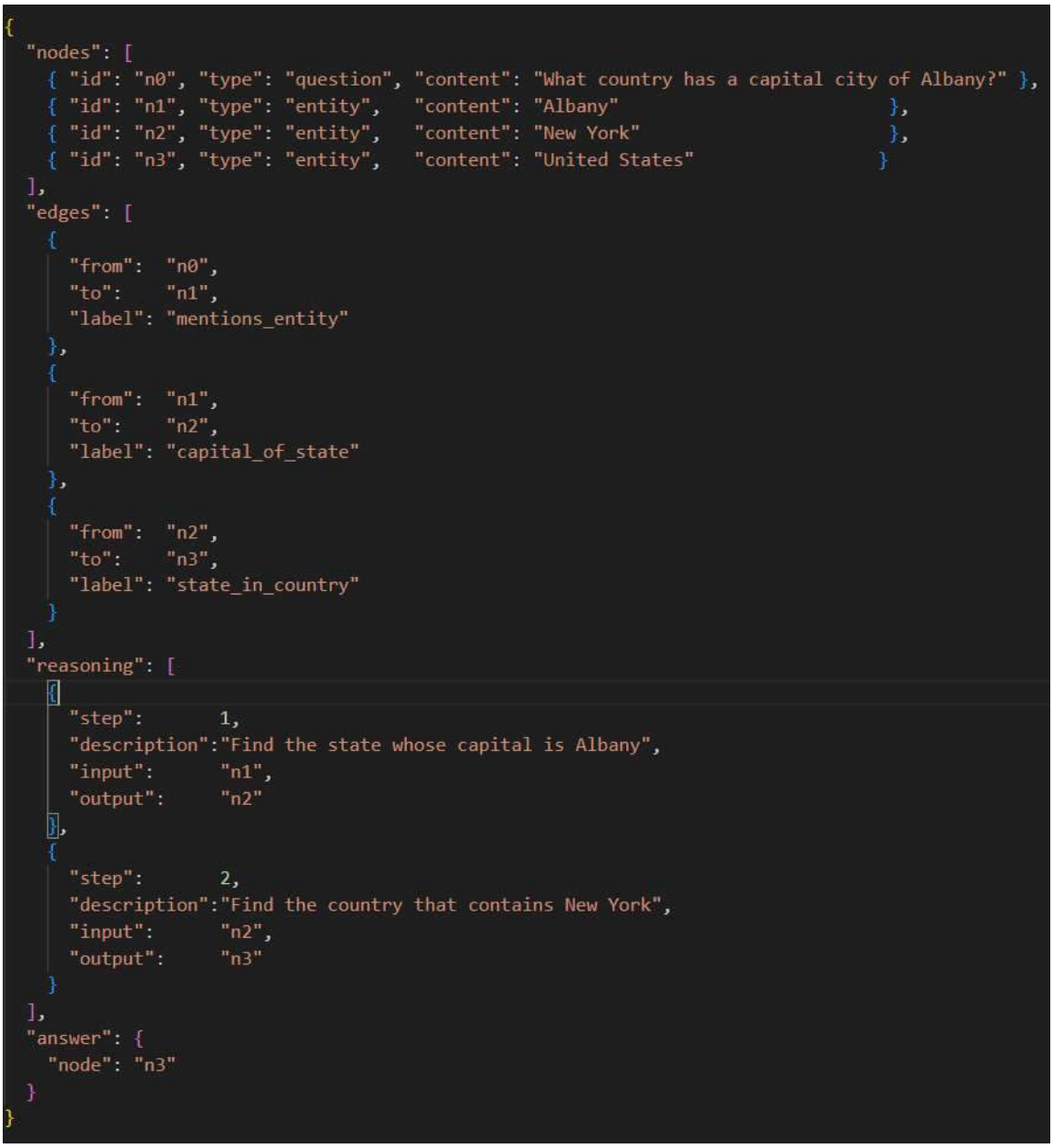
- **Nodes:** Representing questions, intermediate reasoning entities, and final answers.
- **Edges:** Directed connections between nodes labeled clearly to indicate logical or inferential relationships.
- **Reasoning Steps:** An ordered, textual summary of how each inference leads to the next.

The method operates by:

- Feeding identical questions into both a target model and a reference model (Claude Haiku reasoning).
- Capturing the sequential reasoning steps as explicit inference processes.
- Converting these steps into a structured, graph-based JSON format, facilitating both human readability and computational analysis.

3. Model Output Structure to Mimic JSON Graph

To effectively utilize this new method, model outputs must follow a clearly defined, JSON-compatible graph structure:



```
{
  "nodes": [
    { "id": "n0", "type": "question", "content": "What country has a capital city of Albany?" },
    { "id": "n1", "type": "entity",   "content": "Albany"                                      },
    { "id": "n2", "type": "entity",   "content": "New York"                                    },
    { "id": "n3", "type": "entity",   "content": "United States"                              }
  ],
  "edges": [
    {
      "from":  "n0",
      "to":    "n1",
      "label": "mentions_entity"
    },
    {
      "from":  "n1",
      "to":    "n2",
      "label": "capital_of_state"
    },
    {
      "from":  "n2",
      "to":    "n3",
      "label": "state_in_country"
    }
  ],
  "reasoning": [
    {
      "step":       1,
      "description":"Find the state whose capital is Albany",
      "input":      "n1",
      "output":     "n2"
    },
    {
      "step":       2,
      "description":"Find the country that contains New York",
      "input":      "n2",
      "output":     "n3"
    }
  ],
  "answer": {
    "node": "n3"
  }
}
```

This structured output ensures consistency, ease of analysis, and transparency in interpreting model decisions.

4. Comparing Target and Reference Model Graphs for Scoring

The final step involves comparing the attribute graphs generated from the target and reference models to derive an explainability score (0-10). The comparison metrics are:

1. **Node Overlap (20 %)**
    a. Count how many nodes appear in both the target and the reference graphs.
    b. Divide that by the total number of nodes in the reference graph (gives a value between 0 and 1).
    c. Multiply by 2 to get the node contribution.
2. **Edge Overlap (20 %)**
    a. Count how many edges appear in both graphs.
    b. Divide by the total number of edges in the reference graph.
    c. Multiply by 2 to get the edge contribution.
3. **Reasoning Overlap (20 %)**
    a. Count how many reasoning steps align (exact or fuzzy match) between target and reference.
    b. Divide by the total number of reasoning steps in the reference.
    c. Multiply by 2 to get the reasoning contribution.
4. **Final Answer Match (40 %)**
    a. If the two final answers convey the same meaning (even if worded differently), this is 1; otherwise it's 0.
    b. Multiply by 4 to get the answer contribution.

Total Score = (3 × node overlap) + (3 × edge overlap) + (4 × answer match)

Example

- Node overlap = 8 / 10 = 0.8 → 2 × 0.8 = 1.6
- Edge overlap = 12 / 15 = 0.8 → 2 × 0.8 = 1.6
- Reasoning overlap = 1/ 2 = 0.5 → 2 × 0.5 = 1.0
- Final answers match → 4 × 1 = 4.0

Score= 2.4 + 2.4 + 4.0 = round (**8.2) -->8**

A higher score indicates greater alignment between the target model and the reference model, reflecting stronger, clearer, and more transparent reasoning processes.

Business Requirements Document (BRD)

MagicAudit – Default Fileset in Objective Testing

1. Business Context

In the current system, users must select or upload filesets when running an Objective Testing evaluation. This creates friction when running standard compliance checks. To simplify, we introduce a Default Fileset option.

- End User Benefit: Faster evaluations without manual file selection.

- Admin Benefit: Centralized control of filesets per score type.

2. Objectives

- Provide a "Use Default Fileset" option for evaluations in Objective Testing.

- Restrict users from uploading/selecting files when default mode is enabled.

- Enable Admins to upload and configure default filesets per score type.

- Support multiple file formats (text, JSON, audio) based on score type requirements.

- Only one score type allowed at a time.

3. Scope

In-Scope

- Checkbox: "Use Default Fileset" in Objective Testing evaluation form.

- Disable file upload/selection when default option is checked.

- Backend support to link Admin-configured default filesets.

- Admin Configuration UI for uploading files per score type.

- Only one score type allowed at a time.

Out-of-Scope

- Automatic file generation.

- Multi-score evaluations (only one score type allowed at a time).

5. Functional Requirements

5.1 User Workflow (Governance → Objective Testing)

- User selects: Feature + Score Type.

- File Selection Area:

  - Checkbox: "Use Default Fileset".

  - If checked → File upload/selection UI is disabled.

  - System uses Admin-defined default fileset for selected score type.

- Rule: Only 1 Score Type allowed per evaluation.

- User proceeds with evaluation using the default file.

5.2 Admin Workflow (Governance → Admin → Configuration → Objective Test Default Filesets)

- Admin uploads filesets for each score type.

- File type requirements vary:

  - Single Text File (Output) → Absence of Bad Words, Text Quality, AI Explainability, Hallucination (all subtypes), etc.

  - Two Text Files (Output + Reference) → CoSelection, Completeness.

  - Text (Output) + JSON (Reference) → Utility, Full Disclosure.

  - Audio (WAV Input) + Text Output → WECLevel, WordLossLevel.

- UI allows:

  - Dropdown/List of scores: Score Type.

  - File upload fields (dynamic based on score type).

  - Validation: Ensure correct file formats.

  - Save/Update/Delete default fileset.

6. Non-Functional Requirements

- Files should be securely stored and linked to score types.

- Default filesets must be version-controlled (latest version active).

7. UI/UX Requirements

7.1 User Side (Objective Testing)

- Checkbox: “Use Default Fileset” above file selection area.

- If checked:

  - File upload & select buttons are greyed out.

  - Info message: “System will use default fileset configured by Admin for this score type.”

7.2 Admin Side (Configuration → Default Filesets)

- Table of existing score types with uploaded filesets.

- Buttons: Upload / Replace / Delete.

- Status: Configured / Missing

8. Workflow

User

1. Go to Governance → Objective Testing.

2. Select Feature + Score Type.

3. Check “Use Default Fileset.”

4. Run evaluation → system uses default file.

Admin

1. Go to Governance → Admin → Configuration → Objective Test Default Filesets.

2. Select Score Type.

3. Upload required files (based on score type).

4. Save configuration.

5. Default file now available to users.

9. Dependencies

- File storage system.

- Governance module (Objective Testing).

- Admin module (Configuration).

10. Risks & Mitigation

- Risk: Wrong file format uploaded.

  - Mitigation: File type validation at upload.

- Risk: Users bypass default rule.

  - Mitigation: UI + backend restriction enforcement.

- Risk: Missing default file.

  - Mitigation: Warning message: "No default fileset configured. Please upload in Admin section."

11. Acceptance Criteria

- User can enable "Use Default Fileset" and run evaluation without selecting/uploading files.

- Only one score type is allowed when default is enabled.

- Admin can upload and manage filesets for all score types.

- System validates correct file types per score type.

- Evaluation runs successfully with default fileset.

Prediction Report (BRD)

Prediction Report – Future Score Forecasting

Title: Prediction Report – Future Score Forecasting

Version: 1.0

Date: 2025-05-20

Prepared By: [Mahabub Hossain]

1. Overview

This document outlines the requirements for building a Prediction Report feature within magicAudit. The report predicts future performance scores for selected Objective Testing "Score Types", leveraging historical score data and forecasting models (Time Series Algorithm). The goal is to provide forward-looking insights that enable users to anticipate risks, quality trends, and areas requiring intervention.

2. Objective

- Allow users to select a Score Type (e.g., Text Quality, Completeness, No PII, etc.) and request prediction for a defined time range (e.g., next 10 days).
- System analyzes historical scores and generates future score predictions.
- Extendable to other testing categories (e.g., Reflective Testing) in future enhancements.

3. Scope

3.1 In-Scope (MVP)

- Prediction limited to Objective Testing Score Types, including but not limited to:
  - Absence of Bad Words
  - Completeness
  - CoSelection
  - Full Disclosure
  - No Criminality
  - No HateSpeech
  - No SelfHarm
  - No PII
  - No PCI
  - No PHI
  - Text Quality
  - Utility
  - No Violence
  - Word Error Count
  - Word Loss Level

- Hallucination-Related Scores (enhancement – limited due to smaller data availability)
- AI Maturity (enhancement – limited due to smaller data availability)

- Users can select:
  - Score Type
  - Prediction period (e.g., next N days, weeks, or months).

- Report output includes:
  - Predicted score trend for selected period.
  - Confidence interval (optional, based on model support). (Enhancement)
  - Visual representation (chart/graph).

3.2 Out of Scope (MVP)
- Reflective Testing prediction (to be discussed, see Section 9).
- Predictions beyond the available data context (e.g., if only 30 days history exists, user cannot request 1-year predictions).

4. Functional Requirements

4.1 User Capabilities
- Select a Score Type from dropdown list.
- Input prediction duration (e.g., next 10 days).
- View predicted scores in:
  - Graph format (line chart with trend).
- Export prediction results (CSV, XLSX, SVG).

4.2 Prediction Engine Requirements
- Use historical score data stored in system.
- Apply forecasting model (e.g., time series analysis: ARIMA, SARIMA).
- Ensure predictions adapt dynamically to available history length. (Enhancement)
- Handle missing or sparse data gracefully (e.g., fallback to shorter prediction window).

4.3 Validation Rules
- Prediction request must specify:
  - Valid score type.
  - Valid date range (positive integer, within system limits).
- If insufficient data:
  - Display warning ("Not enough historical data to predict future scores for X"). (Enhancement)

5. Non-Functional Requirements

- Performance: Predictions should be generated within 5 seconds for typical queries (up to 1 year history, 30-day prediction).
- Scalability: Must support multiple concurrent prediction requests.
- Accuracy: Baseline forecast accuracy should be measured and periodically recalibrated.

- Security: Role-based access (only authenticated users with project access can generate prediction reports).
- Auditability: Log all prediction requests with user, score type, prediction window, and timestamp.

6. Data Requirements

- Input data: Historical scores per project per score type.
- Output data: Predicted score dataset (date, predicted value, confidence band if supported).
- Storage: Predictions may be generated on-demand (not persisted), but system should allow caching for repeated queries.

7. UI/UX Requirements

- Prediction Report Screen
  - Score Type dropdown.
  - Date range selector (e.g., Next X Days).
  - "Generate Prediction" button.

- Results Section
  - Line chart (historical + predicted scores).
  - Export options.

- Error Handling
  - Inline error messages if prediction cannot be generated (e.g., insufficient data).

8. Security & Access Control

- Only authorized users with access to the project scores can request prediction reports.
- Audit log must capture:
  - User ID
  - Score Type
  - Prediction window
  - Date/Time of request

9. Open Questions / Future Enhancements

1. Reflective Testing Scope:
   - Should predictions also be available for Reflective Testing metrics?
   - If yes, this becomes part of Phase 2 Enhancements.

2. Multi-Score Prediction:
   - Should system allow predicting multiple score types simultaneously (e.g., Text Quality + Utility)?

3. Prediction Accuracy Transparency:
   - Should system expose confidence intervals to end users?

4. Alerts/Thresholds:
   - Should predicted drops in score trigger proactive alerts?

10. Risks & Mitigations

Risk	Impact	Mitigation
Insufficient historical data	Predictions may be unreliable	Restrict prediction requests to minimum history length
Over-reliance on forecast	Misinterpretation by users	Provide disclaimer: "Predictions are estimates based on past trends."
Performance lag for large datasets	Poor user experience	Implement caching & optimized time-series models
Data quality issues	Skewed predictions	Data validation & cleansing before modeling

11. Glossary

Prediction Report – A report that forecasts future values of a selected score type.
Score Type – A measurable dimension from Objective Testing (e.g., Text Quality, Utility).
Forecasting Model – Statistical or machine learning method used to predict future scores.
Confidence Interval – Range of values within which the predicted score is expected to fall with a given probability.

Diagnostic & Prescriptive Report (BRD)

Feature Name: Diagnostic & Prescriptive Report

Module: Evaluation / Scoring Insights

Version: 1.0

Owner: [Your Name / Team]

Date: August 8, 2025

Part 1: Diagnostic Model (Current Implementation)

1. Objective

To allow users to ask questions related to scoring dimensions (e.g., Text Quality, Full Disclosure, Utility), and receive only error classifications from the diagnostic model.

2. Scope

Accepts user questions and returns only error classification. No model-generated answers or suggestions.

3. Error Classifications

Error Type	Description
Factual Error	Inaccuracy or misrepresentation of facts
Risk Error	Implies risky behavior or non-compliance
Clarity Error	Vague or ambiguous question
Relevance Error	Off-topic or unrelated to the score

4. Functional Requirements

- DR-001: Input field for user questions
- DR-002: Dropdown to select score context (Not Necessary)
- DR-003: Submit question to diagnostic model API
- DR-004: Display error classification result
- DR-005: Maintain question history
- DR-006: Support multiple sessions
- DR-007: Provide helper text/tooltips for guidance

5. Non-Functional Requirements

- Usability: Chat-like interface
- Performance: Response within 2–4 seconds
- Security: No PII submitted
- Scalability: Support multiple users
- Reliability: Always return a valid error type or fallback message

6. UI Components

- Question Input Box
- Score Context Dropdown (Not Necessary)
- Submit Button
- Error Classification Badge (e.g., "Factual Error")
- History Panel
- Helper Tooltip

7. API Requirements

- DR-API-001: Endpoint to accept question + score context
- DR-API-002: Response includes: error_type
- DR-API-003: Log questions and classifications (anonymized)
- DR-API-004: Fallback for unclassified errors

Part 2: Prescriptive Analysis (Future Enhancement)

1. Objective

To provide prescriptive suggestions and resource links based on the error classification returned by the diagnostic model.

2. Scope

Takes error classification as input and returns:
- Prescriptive suggestion
- Optional links to documentation, examples, or templates

3. Functional Requirements

- PA-001: Accept error classification from diagnostic model
- PA-002: Generate prescriptive suggestion text
- PA-003: Provide links to relevant resources
- PA-004: Display suggestions in same UI section (future merge)
- PA-005: Allow user feedback on suggestions for improvement

4. UI Enhancements (Future)

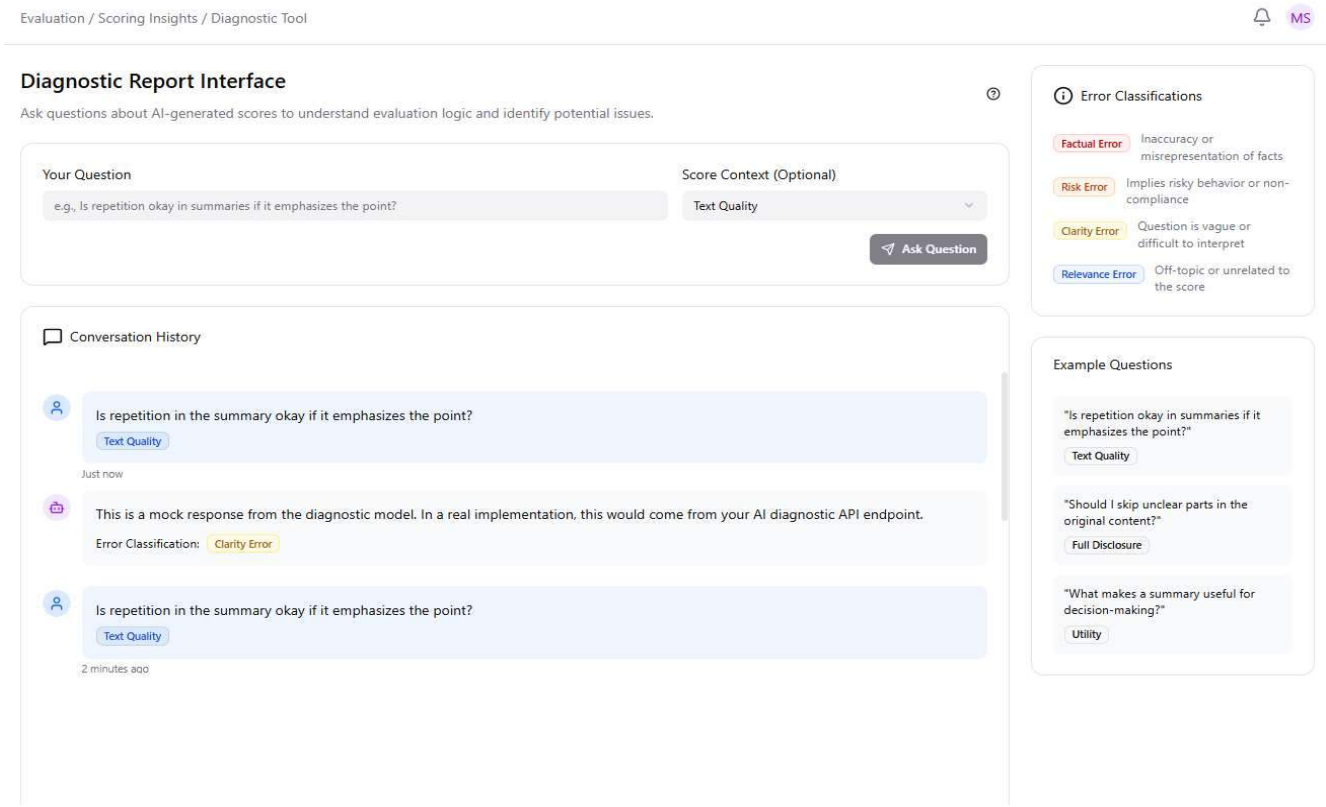
- Prescriptive Suggestion Panel below diagnostic response
- Resource Links (e.g., writing guides, templates)

- Feedback Button (“Was this helpful?”)

5. Example Extension

Error Type	Prescriptive Suggestion	Resource Link
Factual Error	Verify facts using trusted sources	[Link to style guide]
Risk Error	Avoid skipping content; summarize instead	[Link to summarization tips]
Clarity Error	Rephrase question for clarity	[Link to question framing guide]

***Sample design:



Business Requirement Document (BRD) AI Maturity Tree – Weightage and Structure Management

Title: AI Maturity Tree – Weightage and Structure Management

Version: 1.0

Date: 2025-05-15

Prepared By: [Mahabub Hossain]

1. Overview

This document outlines the requirements for managing an AI Maturity Tree, used to evaluate and track the maturity level of AI implementations within magicAudit. The maturity tree consists of nested hierarchical branches, each with leaf nodes that are assigned weightage values. This system supports customization of weightage and, in future enhancements, the structural modification of the tree itself.

2. Objective

To provide an intuitive and controlled interface for users to:
1. Part A (MVP Scope): Edit the weightage values of leaf nodes within a fixed tree structure.
2. Part B (Future Enhancements): Enable users to modify the tree structure itself, including adding/removing/moving nodes, while preserving system constraints.

3. Current Tree Structure (Baseline)

Type
├── Build Types
│ ├── Foundational Model
│ │ ├── LLM Hosted by Large Providers
│ │ │ ├── Prompt Engineering – 5
│ │ │ ├── RAG – 10
│ │ │ ├── RAFT – 15

```
│   │   │   ├── RLHF – 20
│   │   │   ├── Agents – 25
│   │   │   └── Locally hosted small LLM – 25
│   ├── Custom Built classic AI model
│   │   ├── POC Models – 45
│   │   └── Deployed Models – 55
├── Evaluation Types
│   ├── AI Evaluation Systems Not Implemented – 1
│   ├── Partial Implementation – 39
│   └── Fully Implemented Evaluation System – 60
└── MLOps
    ├── Not Implemented – 1
    ├── Partially Implemented – 39
    └── Fully Implemented – 60
```

4. Part A: Weightage Management (MVP)

4.1 Functional Requirements

- Users should be able to edit the weightage values of leaf nodes.
- Changes should be restricted to leaf nodes only.
- The sum of all leaf nodes under a single parent (branch) must equal 100.
- No leaf node value can be 0 or null.
- The tree structure is read-only in this phase.
- Any violation of constraints should trigger validation errors with appropriate messages.

4.2 Non-Functional Requirements

- Interface must provide real-time validation.
- Weightages should be persisted securely and version-controlled.
- API support for fetching/updating weightages.
- Audit logs for tracking changes in weightage values.

4.3 Constraints

- Fixed tree structure in this phase.
- Admin-level access is required to perform updates.
- The system should block submission unless constraints are met.

5. Part B: Future Enhancements – Tree Structure Modification

5.1 Functional Requirements

- Users should be able to:
  - Add new nodes (branches or leaves).
  - Modify hierarchy (move nodes to new branches).
  - Rename existing nodes.
  - Delete nodes (with confirmation & impact analysis).
- Modifications should be allowed only under admin supervision.

5.1.1 Detailed Functional Requirements

1. **Node Creation**
   a. **Branch Nodes**
      i. Allow creation of new parent/branch nodes at any level except under leaf nodes.
      ii. Require a unique name within the same parent branch.
      iii. No weightage assigned to branches directly; weightage applies only to leaves.
   b. **Leaf Nodes**
      i. Must be created under a valid branch node.
      ii. Require a name and initial weightage.
      iii. Weightage must respect the sum rule (total 100 under parent).
2. **Node Modification**
   a. **Rename**
      i. Ability to rename branch or leaf nodes.
      ii. Renaming should not affect existing historical versions or reporting references — unique IDs remain unchanged.
   b. **Move Node**
      i. Move a node (branch or leaf) to a different parent branch using drag-and-drop or a "Move to" action.
      ii. Automatically triggers revalidation of weightage sum rules for both source and target branches.
3. **Node Deletion**
   a. **Branches**
      i. Can only be deleted if they have no child nodes, or if deletion cascades (with explicit user confirmation).
      ii. System should display the impact of deletion (e.g., loss of related scores or historical data references).
   b. **Leaves**

        i. Deleting a leaf automatically requires rebalancing the weightage for remaining leaves under the same branch.

4. **Version Management (Optional)**
    a. Every structural change should create a **new version** of the maturity tree.
    b. Each version should have metadata:
        i. Version number
        ii. Created by (User ID)
        iii. Date/time
        iv. Change summary
    c. Ability to **preview previous versions.**
    d. Ability to **rollback** to a previous version with confirmation.
5. **Validation Engine Enhancements**
    a. Must validate:
        i. Weightage sum = 100 for all leaf sets under a branch.
        ii. No duplicate names under the same parent branch.
        iii. No leaf node with null/zero weightage.
        iv. No orphan nodes (every node must have a valid parent except the root).

5.1.2 Suggested UI/UX Requirements

1. **Tree Builder Interface**
    a. Drag-and-drop node placement.
    b. Context menu on each node: Add, Rename, Move, Delete.
    c. Expand/collapse branches for large trees.
2. **Weightage Input Controls**
    a. Numeric input or slider for leaf weightage.
    b. Live total counter showing sum of sibling leaf weightages.
    c. Inline error messages when constraints are violated.
3. **Impact Analysis Dialog**
    a. For deletion or move actions, display:
        i. Affected branches/leaves.
        ii. Any scoring impacts.
        iii. Recommendation for rebalancing weightages.
4. **Preview & Publish Flow**
    a. Changes saved in a **draft mode** until validated and published.
    b. Preview mode to see the modified tree alongside the current live version.
    c. "Publish" action requires passing all validation checks.
5. **Version History Screen (Optional)**
    a. List all past versions with change summaries.
    b. Option to restore or clone a version.

5.2 Structural Rules & Constraints

- Sum of leaf nodes under any branch must equal 100.
- No leaf node weightage can be 0 or null.
- A leaf cannot exist without a valid parent branch.
- Structural modifications must pass a validation engine before being saved.

5.3 Suggested UI Capabilities

- Tree builder interface (drag-and-drop or visual layout).
- Weightage slider/input for each leaf node with sum validator.
- Pre-publish validation and preview mode.
- Version history and rollback option for previous tree structures.

6. Security & Access Control

- Role-based access to ensure only authorized users can:
  - Modify weightage values (Part A).
  - Edit tree structure (Part B).
- Logs must capture:
  - User ID
  - Date/time
  - Action type (create/edit/delete)
  - Affected nodes and weightages

7. Glossary

Term	Description
Leaf Node	A terminal node (cannot have children), holds weightage.
Branch/Parent Node	A node that can have one or more children.
Weightage	A numerical value contributing to maturity

	assessment.
Validation Engine	System enforcing sum rules and non-null requirements.

8. Appendix

Example of Constraint Violation:

If user attempts:
LLM Hosted by Large Providers:
Prompt Engineering – 0
RAG – 10
RAFT – 20
RLHF – 30
Agents – 40
Locally hosted small LLM – null

System Response:
❌ "Prompt Engineering cannot be 0. Please assign a non-zero value."
❌ "Locally hosted small LLM cannot be null."
❌ "Sum of values is 100, but includes invalid entries."

Business Requirement Document (BRD)

Feature: Diagnostic Report

Module: Evaluation / Scoring Insights

1. Objective

To provide users with a diagnostic interface where they can ask model-related questions concerning AI-generated scores (e.g., Text Quality, Full Disclosure, Utility), and receive interpretable, categorized feedback from a diagnostic LLM model.

This feature aims to:
- Improve transparency of scoring logic
- Support SMEs, reviewers, or analysts in understanding why a score was applied
- Highlight specific error categories linked to the user's query

2. Background / Context

AI-generated content on the platform is evaluated using scores such as:
- Text Quality
- Full Disclosure
- Utility
- Absence of Bad Words
- ...etc.

To increase transparency and usability, users can now engage with a diagnostic model by asking custom questions related to these scores. The model responds with:
- A natural language answer
- A classified error type, if applicable

3. Error Classifications

Error Type	Description
Factual Error	Inaccuracy or misrepresentation of facts in the question or underlying assumption
Risk Error	The question reflects or implies risky behavior or non-compliance
Clarity Error	The question is vague, ambiguous, or difficult to interpret clearly
Relevance Error	The question is off-topic or unrelated to the

	specific score being discussed

4. Functional Requirements

Requirement ID	Description
DR-001	Provide an input field where users can type their diagnostic question related to a score.
DR-002	Allow selection (or pre-selection) of the score context (e.g., Text Quality, Utility) that the question refers to.
DR-003	On submission, pass the question and selected score to the diagnostic model API.
DR-004	Display the model's response text and the corresponding error classification (e.g., Factual Error).
DR-005	Maintain a conversation history panel showing previously asked questions and answers.
DR-006	Support multiple diagnostic sessions without refreshing the page.
DR-007	Add helper text explaining the purpose of this tool and how to frame questions effectively.
DR-008	Display a contextual helper message above the configuration form explaining that updates will impact Auto Optimize behavior across the platform.

5. Non-Functional Requirements

- Usability: Clean, chat-like interface to encourage interaction
- Performance: Response from the diagnostic model should be returned within 2–4 seconds
- Security: No personally identifiable information (PII) should be submitted
- Scalability: The system should support simultaneous use by multiple users without degradation
- Reliability: Responses should always include a valid error type or a message indicating uncertainty

6. UI/UX Overview

UI Components:
- Question Input Box: User types a natural-language question
- Score Context Dropdown: (Optional) Select score type
- Submit Button
- Model Response Panel: Answer text and Error classification tag
- History Panel: Shows past Q&A interactions in reverse chronological order
- Helper Tooltip or Info Icon: “Ask score-related questions (e.g., 'Is repetition okay in summaries?'). Model responses will include explanations and relevant error types to help clarify evaluation logic.”

7. API/Model Integration Requirements

Requirement ID	Description
DR-API-001	Expose an API endpoint to accept a question and optional score context.
DR-API-002	Model response must include: answer, error_type, and optional metadata.
DR-API-003	API must log questions/answers for audit and fine-tuning (with anonymization).
DR-API-004	Include fallback handling if the model cannot classify the error confidently (e.g., “Unclear – No error type assigned”).

8. Example Interactions

User Question	Score Context	Model Answer	Error Classification
Is repetition in the summary okay if it emphasizes the point?	Text Quality	Repetition can harm clarity unless it's strategic.	Factual Error
If some parts are unclear in the original, I can just skip them?	Full Disclosure	Skipping unclear parts risks missing important info.	Risk Error
Is there a standard template of questions used for this score?	Utility	Templates vary by context; no universal one exists.	Clarity Error

9. Future Enhancements

- Multi-question support per score for deeper diagnostics
- Highlight text from the original output related to the diagnostic answer
- Fine-tune diagnostic model based on frequently asked questions
- Allow exporting of diagnostic history

Business Requirement Document (BRD)

Module: Admin → Specific Settings → SME Config → Auto Optimize (LLM Config Management)

1. Objective

Enable Admin users to configure which LLM provider and model is used by the Auto Optimize functionality. The interface will allow users to:
- Select the LLM provider (api_type)
- Choose a model available under that provider
- Input or update the API key
- Save and apply the configuration

2. Current State

The system uses a static config file path defined as:
"OAI_CONFIG_LIST_PATH": "Configs/OAI_CONFIG_LIST.json"

This file contains a list of LLM configurations, for example:

```
[
 {
  "model": "gemini-1.5-pro-001",
  "api_key": "AIzaSyDHiw...",
  "api_type": "google"
 }
]
```

The system (Autogen) reads from this list, but currently lacks a UI for managing or editing it.

3. Proposed Enhancement

Provide a UI under Admin → Specific Settings → SME Config → Auto Optimize, allowing users to:
- Select an API Type (LLM Provider) from a dropdown (e.g., google, openai, anthropic)
- View and select a model based on the chosen API type
- Enter or update the API key
- Save and apply the configuration to update OAI_CONFIG_LIST.json

Note: While the config file supports multiple LLM entries, the Admin UI will only allow selecting and updating one active configuration to be used by Auto Optimize.

4. Functional Requirements

Requirement ID	Description
SME-AO-001	Admin selects an API Type (api_type) from a dropdown list.
SME-AO-002	Upon selection of an API Type, the Model dropdown is dynamically populated with available models for that provider.
SME-AO-003	Admin can then enter or update the API key corresponding to the selected model.
SME-AO-004	API key field must support visibility toggle (show/hide).
SME-AO-005	On Save, the config should be written to OAI_CONFIG_LIST.json, replacing or updating the targeted entry.
SME-AO-006	Auto Optimize functionality must read from the updated config immediately.
SME-AO-007	Validation: All fields (api_type, model, api_key) are required.
SME-AO-008	Display a contextual helper message above the configuration form to explain the purpose of the setting — specifically, that any changes made to the LLM provider, model, or API key will immediately influence the behavior of Auto Optimize across the platform.

5. Non-Functional Requirements

- Security: Mask API key input in UI and store it securely.
- Usability: Responsive dropdowns and error messaging for missing fields.
- Performance: No noticeable lag or downtime after saving new config.
- Access Control: Only users with Admin-level privileges can modify the config.

6. UI Interaction Flow

Step 1: Select LLM Provider

- Dropdown labeled “LLM Provider (API Type)”
- Options: google, openai, anthropic, etc.

Step 2: Select Model

- Based on provider, show dropdown of models (fetched dynamically or pre-defined mapping)

Step 3: Enter API Key
- Text input with show/hide toggle

Step 4: Save or Cancel
- Save: Update the OAI_CONFIG_LIST.json file
- Cancel: Discard changes

7. Backend/API Requirements

Requirement ID	Description
SME-AO-BE-001	Read the full list of LLM configurations from OAI_CONFIG_LIST.json.
SME-AO-BE-002	On update, either replace the specific entry (e.g., by api_type/model) or mark one (If multiple entries are there) as active.
SME-AO-BE-003	Ensure existing configurations in the list are not overwritten unless intentionally modified.
SME-AO-BE-004	Autogen should continue consuming this list, and use the configuration marked or placed as "active".

8. Future Enhancements

- Add API key testing before save
- Versioning and rollback of config
- Config history/audit logs

Business Requirement Document (BRD)

Feature: Enhanced Scheduler – Recurring Event System

Author: Mahabub Hossain

Date: July 15, 2025

Version: 1.0

1. Purpose

This document outlines the business requirements for upgrading the existing scheduler functionality in magicAudit to support flexible and complex recurring schedules, similar to Microsoft Teams' calendar system. This enhancement applies to scheduling Objective Evaluations and Reflective Interviews and will allow users to configure recurring sessions with maximum customization.

2. Background

Currently, magicAudit supports:
- One-time scheduling (date & time selection)
- Basic recurring logic:
  - User sets a start date/time and end date
  - System triggers evaluation/interview every 24 hours within that range

However, this lacks flexibility. Users often require patterns like:
- Repeat every 2 days
- Repeat weekly on specific days (e.g., Mon, Wed, Fri)
- Repeat every month on the 15th
- Indefinite repetitions until manually stopped

3. Objectives

- Implement a robust, customizable recurrence scheduler
- Provide UI/UX experience similar to Microsoft Teams' repeat menu
- Allow scheduling for both Objective Testing and Reflective Interview
- Offer seamless rescheduling of recurring instances
- Ensure proper backend trigger/job handling for each recurrence

4. Functional Requirements

ID	Requirement	Description

FR1	One-Time Scheduling	Continue support for single, specific date/time events. (UTC Time)
FR2	Recurring Scheduling – Frequency	Allow user to choose repeat frequency: Daily, Weekly, Monthly, Yearly.
FR3	Recurring Scheduling – Interval	Let users define intervals (e.g., every 2 days, every 3 weeks).
FR4	Day Selection (Weekly)	Users can select multiple days in a week (e.g., Tue, Thu).
FR5	End Conditions	Users must define end condition: no end, end on date, or end after X occurrences.
FR6	UI Design	Provide an intuitive and collapsible UI panel (inspired by Teams) for configuring recurrence.
FR7	Time Zone Support	All time inputs and outputs will be in UTC. No local conversion supported.
FR8	Update/Cancel Recurrence	Support updating/deleting specific occurrences or the entire series. (Out of scope for now, needs discussion)
FR9	Preview Summary	Show human-readable text summary of recurrence pattern.

5. Non-Functional Requirements

ID	Requirement	Description
NFR1	Reliability	Jobs must trigger reliably as per schedule without missing time slots.
NFR2	Scalability	Must handle high-volume of scheduled tasks across multiple users and clients.
NFR3	Usability	UI must remain intuitive, clean, and responsive.
NFR4	Error Handling	System must gracefully handle scheduling failures or conflicts.
NFR5	Time Standardization	All times are strictly in UTC; no local time conversion will be supported.

6. UI/UX Requirements

New UI Panel: Modal/Section (UI behavior inspired by Microsoft Teams)

- Start: Date Picker + Time Picker (Choose initial date/time)
- Repeat: Dropdown (Frequency options: Daily, Weekly, Monthly, Yearly)
- Interval: Input Box (Define repeat interval, e.g., every 2 days)
- Days: Multi-select (Select days like Mon, Tue, etc. for weekly repeat)
- End Condition: Radio Buttons (No End, End After [x] Occurrences, End By [date])
- Summary: Display Text (Auto-updating line like: "Repeats every Tuesday until Jan 6, 2026")
- Save: Button (Save recurrence configuration)

7. Backend Considerations

- Store recurrence as RRULE or structured JSON for flexibility.
- Integrate with a job scheduler (e.g., Celery, APScheduler, cron) to execute tasks at defined times.
- All time inputs and outputs will be in UTC.
- No conversion to local time zones required or supported.
- Implement logic for recurrence conflict detection (e.g., overlapping evaluations).

8. Risks & Mitigations

Risk	Impact	Mitigation
Job Overload	High	Use queueing/batching for job executions.
Timezone Bugs	Low	All time handling is in UTC, minimizing complexity.
Misconfigured Repeat	Low	Add preview summary and clear UI to reduce confusion.

9. Future Enhancements

- Calendar View Integration: Show all upcoming events in a full-page calendar.
- Bulk Actions: Pause/resume/Edit/delete recurring series from dashboard.
- Notification System: Reminders before each evaluation/interview.
- Integration Hooks: Auto-sync with Microsoft Teams/Google Calendar.

Logs & Auditing for magicAudit (BRD)

Feature: Logs & Auditing for magicAudit

Author: Mahabub Hossain

Date: July 14, 2025

Version: 1.0

1. Purpose

This document outlines the requirements for implementing a centralized Logs & Auditing system in the magicAudit platform. The feature aims to capture all user activities, backend errors, and login/logout-related history. It also provides robust filtering and exporting capabilities to support operational audits, troubleshooting, and compliance needs.

2. Scope

In Scope

- Logging of all client-specific activities within the application
- Capturing backend/server-side error logs
- Tracking user login/logout attempts
- Filter and search functionality for all logs
- Export logs to PDF and Excel
- UI views for activity logs, error logs, and login history

Out of Scope

- Real-time alerting or anomaly detection
- Deep forensic audit tools (e.g., user session replay)

3. Functional Requirements

ID	Requirement	Description
FR1	Activity Logs	Capture all interactions (Status, Details etc) with the MagicAudit application by users within a client.
FR2	Error Logs	Log all backend errors (e.g., API failures, exceptions,

		Status, Error Codes, Details) scoped per client.
FR3	Login History	Track all login/logout events, including failed login attempts and timestamps, Status, Details etc.
FR4	Filter Options	Provide filters on all log views by: user, date range, domain (Governance, SME & Studio), log type, Status, Details etc.
FR5	Export Logs	Enable users to export filtered logs in Excel and PDF format.
FR6	Search Logs	Support keyword search across logs for faster lookup.
FR7	Secure Access	Only authorized roles can access log and audit sections.
FR8	Data Retention	Define a log retention policy (e.g., retain logs for 1 year by default).
FR9	Timestamping	All logs must include UTC timestamps UI.

4. Non-Functional Requirements

ID	Requirement	Description
NFR1	Performance	Logs must be retrievable and filterable efficiently, even with high volumes of data.
NFR2	Security	Sensitive data in logs must

		be masked or encrypted. Access must be audit-tracked.
NFR3	Compliance	Ensure logs meet basic audit trail requirements for client-level transparency.
NFR4	Availability	Logs should be available even if parts of the system (e.g., APIs) fail.
NFR5	Timezone Awareness	Display & store timestamps in UTC.

5. UI Requirements

Section	UI Element	Details
Logs Dashboard	Tab View	Tabs: Activity Logs, Error Logs, Login History
Filter Panel	Multi-select Filters	User, Domain, Date Range, Log Type
Table View	Paginated Log Table	Columns: Timestamp, Action/Event, User, Domain, Status, Error Details
Export Button	Download	Buttons to export visible logs as PDF or Excel
Search Bar	Full-text Search	Search by event, error keyword, or user email

6. Log Types Overview

Type	Source	Example Events
Activity Log	User/Application	API triggered, project config updated, LLM key changed
Error Log	Backend Services	API timeout, database error,

		unhandled exception
Login History	Auth Service	Successful login, logout, failed attempt, session expired

7. Export & Filtering Rules

- Filters: Applied across all logs; persistent across session
- Export Options: PDF and Excel formats
- Export Scope: Full dataset or filtered view
- Role Access: Export only for admins/auditors
- Max Export Range: Limit to 10,000 records per download for performance

8. Risks & Mitigations

Risk	Impact	Mitigation
Log overload	Medium	Archive older logs periodically
Unauthorized access to sensitive logs	High	Role-based access control + masking
Slow export performance	Medium	Paginated data + export limits
Timezone confusion	Low	Always store in UTC

9. Future Enhancements

- Real-time error alerting via Slack/Email
- Scheduled reports for compliance audits
- **Action-Level Granularity**

- **Enhancement:** Capture detailed action logs like:
    - "User X updated LLM key for OpenAI"
    - "User Y exported report for Domain: SME"

- **Why:** Adds traceability and supports compliance/audit tracking.

-User Agent & IP Logging

- **Enhancement:** Capture user's **IP address**, **browser**, **device**, and **geo-location (approx)** during login and activity.

- **Why:** Helps detect unusual access patterns and adds forensic traceability.

LLM Model Request Workflow (BRD)

Feature: LLM Model Request Workflow

Module: Admin → General Settings → Model Score Mapping

Application: magicAudit

1. Objective

To replace the current 'Add Model' functionality (Available only in Figma design) with a request-based workflow that allows users to submit requests for adding new LLM models. These requests will be tracked and processed by the development team, and upon completion, the models will be available in relevant dropdowns and settings sections.

2. Stakeholders

---

3. Current Design (To be replaced)

Clicking 'Add Model' opens a popup with:
- Name
- API Key
- Feature selection

Model is added directly and appears in the dropdown.

4. New Requirement Summary

Replace the direct model addition with a model request form and an approval-processing-tracking mechanism.

5. Functional Requirements

5.1. Request Model Addition UI

On clicking 'Add Model', open a form-based popup/page with following fields:

Field	Type	Validation / Notes
LLM Name	Text	Required
Model Name	Text	Required
API Key	Text	Required
Features	Multi-select dropdown	Select or add custom tags

Created By	Auto-filled (Logged-in user)	Non-editable
Created Date	Auto-filled (Today)	Non-editable
ETA	Date Picker	Must be future date only, minimum +1 day from today
Status	Dropdown (default: Pending)	Controlled by backend, user can't edit
Priority	Dropdown (Low, Medium, High)	Required

✅ Add Confirm and Cancel buttons.

✅ On Confirm, the request is submitted.

5.2. Request Submission

- Save the request in the system (DB).
- Send email notification to the assigned Dev Team. (Need to discuss the flow)
- Display success toast/message to user.

5.3. My Requests Section

Create a 'My Requests' table listing submitted requests with columns:

LLM Name, Model Name, API Key, Created Date, ETA, Priority, Last Updated, Status, Feedback.

✅ Status & Last Updated field must update in real-time.

✅ On status update to 'Done', trigger email notification to the requester.

5.4. Developer Dashboard (Future Implementation, Need to discuss the flow)

- View requests assigned to them.
- Allow updating request status.
- Each status update should:
  - Be recorded with timestamp
  - Trigger UI update
  - Notify requester if 'Done' or 'Rejected'

-If Status is rejected then Feedback from Dev team.

5.5. LLM Availability Post Completion

Once status is Done:

- Model appears in LLM dropdowns in Model Score Mapping.
- Model appears in - API Key Management → LLM List.

6. Non-Functional Requirements

Requirement	Description
Real-Time Status Update	WebSocket or polling-based update on UI
Role-Based Access Control	Only logged-in Admin can submit requests
Email Notifications	Must support transactional emails (user & dev)
Data Validation	ETA must not allow today or past dates
UX Consistency	Should follow existing magicAudit UI/UX patterns
Log	Audit logs must track all actions.

7. Email Notification Triggers

Trigger	Recipient	Message Summary
New Request Submitted	Dev Team	New LLM request added by user
Status Updated (Done/Rejected)	Requester	Status of your request is updated

8. UI Impact Summary

Page	Impact Description
Model Score Mapping	1. "Add Model" becomes "Request New Model" 2. Once Done, dropdown updated with newly created LLMs
New Popup	Full form with all new fields
My Requests Section	New tab/page to track model requests
API Key Management	LLM List updated with newly created LLMs

Business Requirement Document (BRD)

Module: Admin Configuration Management

Section 1: Objective Testing

1. Objective

Enable administrators to manage and configure threshold-based scoring logic for objective compliance categories including PII, PHI, PCI, Absence of Bad Words, and audio-based configurations such as WEC, WLL, and No Audio Pause. This ensures transparent evaluation criteria and adaptability across use cases.

2. Stakeholders

- Product Owner
- Admin Users
- Development Team
- QA Team
- Compliance & Security Team

3. Functional Requirements

3.1 Score Selection

A dropdown will be available for the admin to select from:
- No PII
- No PHI
- No PCI
- Absence of Bad Words
- WLL
- No Audio Pause

3.2 Configuration Display & Editability

A. No PII / No PHI / No PCI

These configurations control how the system scores the presence of sensitive personal, health, and payment-related information. Each parameter has a defined weight contributing to the overall risk score.

Admin Capabilities:
- Edit weight values
- Add new parameters
- Remove existing parameters

Example Data:

Parameter	Type	Weight
Email	Number	2
Contact Number	Number	2
Social Security Number	Number	3
Parameter	**Type**	**Weight**
Disease	Number	2
Test Results	Number	2
Doctor's Name	Number	1
Parameter	**Type**	**Weight**
Credit Card Number	Number	2
OTP	Number	3
CIBIL Score	Number	1

B. Absence of Bad Words

This configuration ensures the system flags offensive or inappropriate language. All terms listed are considered prohibited, impacting the score if present.

Admin Capabilities:
- Add new words
- Remove existing words

Data Source: Stored in profanity_wordlist 1.txt as a flat text list.

Example Terms:
anal, asshole, bitch, cumshot, fuck...

C. WLL (Audio Silence Detection)

These parameters define how the system detects natural pauses or silent gaps in audio. This helps in identifying potential delivery issues like unnatural pauses or speaking patterns in audio-based responses.

Admin Capabilities:
- Modify threshold values
- Tune silence and buffer detection

Parameter	Type	Description
minsilencelen	Number	Minimum silence length in ms
silencethresh	Number	Amplitude threshold (can be negative)
buffer	Decimal	Additional margin; Max value = 1

D. No Audio Pause

Used to ensure continuity in speaking. Absence of expected pauses could indicate script reading or unnatural speech flow. These thresholds help detect such cases.

Admin Capabilities:

• Adjust silence detection values

Parameter	Type	Description
minsilencelenNopause	Number	Minimum required silence length
silencethreshNopause	Number	Amplitude threshold (can be negative)

3.3 Save Functionality

• All configuration changes are:
- Validated for data types and constraints (e.g., decimal max 1.0)
- Saved back into JSON or flat file storage
- Logged for auditing and version control

4. Non-Functional Requirements

• Real-time UI validations and alerts
• Bulk editing support for bad words
• Performance optimized for 1000+ config entries
• Audit trails for accountability

5. Future Enhancements

• Categorize bad words by severity/type
• Add preview functionality for sample scoring outcomes
• Enable per-topic configurations for PII/PHI/PCI and audio models

Section 2: Reflective Testing

1. Objective

Provide a unified similarity threshold setting for evaluating reflective interviews. This ensures consistent semantic and contextual similarity scoring across all topics.

2. Configuration Parameters

Parameter	Type	Description
thresholdSimilarity	Decimal	Max value = 1.0

3. Save and Validation

• Accepts decimals up to 1.0
• Stored in central config
• Consumed by evaluation engine during reflective scoring

Section 3: Investigative Testing

1. Objective

Define standard summary length expectations for investigative SME responses. Supports uniformity in how response summaries are measured and compared.

2. Configuration Parameters

Parameter	Type	Description
SummaryLength	Number	Default is 50 (word count)

3. Save and Validation

- Positive integer required
- Stored in investigative config file
- Referenced during investigative review processing

Business Requirement Document (BRD)

Feature: LLM Configuration Management Interface (API Key Management for Existing Models (Gen AI))

Author: Mahabub Hossain

Date: July 2, 2025

Version: 1.0

1. Purpose

The purpose of this document is to define the business requirements for implementing an interface within the magicAudit application that enables users to view, edit, delete, and update API keys and model configurations for various integrated Large Language Models (LLMs) such as OpenAI, Gemini, Claude, Mistral, Anthropic, and LLaMA2.

This is necessary to support flexible switching between providers, updating expired keys, and aligning with performance or cost-based model upgrades.

2. Scope

In Scope

- UI for listing current LLM configurations
- View masked API keys
- Update API keys
- Select models from a dropdown (fetched from DB)
- Delete API key configurations
- Add new API key-model pairs
- Audit log integration (optional enhancement)

Out of Scope

- Direct testing of LLM responses
- Editing of local model paths (like LLaMA2) via UI

3. Functional Requirements

ID	Requirement	Description
FR1	View Configurations	Display all configured LLM providers with their

		associated masked API keys and selected models.
FR2	Edit API Key	Allow users to update an existing API key. The key will be masked by default (****1234), but can be toggled to view temporarily.
FR3	Edit Model	Users can select from a predefined list of models (retrieved from database) associated with each provider.
FR4	Delete Configuration	Users can delete an LLM provider configuration after confirmation.
FR5	Add New Configuration	Users can add a new API key and select a model from the dropdown list for any supported LLM.
FR6	Save to Config	Upon save, update the application's backend configuration store (JSON/DB/etc.).
FR7	Input Validation	Ensure API key format and model selection are validated before submission.
FR8	Access Control	Only admin or privileged users can view or modify configurations.
FR9	Audit Trail (Optional)	Log each update or deletion for traceability and security compliance.

4. Non-Functional Requirements

ID	Requirement	Description
NFR1	Security	All API keys must be encrypted in storage and masked in UI display.
NFR2	Responsiveness	UI must be responsive across devices (desktop/tablet).
NFR3	Role-Based Access	Only authorized roles should have access to edit/view the LLM config.
NFR4	Auditability	Maintain logs of configuration changes (who, when, what).
NFR5	Scalability	Should allow support for more LLM providers in the future.

5. User Interface Requirements

Section	UI Component	Details
LLM List Page	Table View	Shows: Provider Name, Masked API Key, Model, Action Buttons (Edit, Delete, View)
Edit Modal	Form	- API Key (editable input) - Model (dropdown populated from DB) - Save/Cancel buttons
Add New	Button & Form	Similar to Edit Modal
View Key	Button/Icon	Toggles masking of API key
Delete	Confirmation Dialog	Are you sure? with provider name for safety

Role Check	Access Restriction	Hide page if user lacks permission

6. Future Enhancements

- Add expiry alerts for API keys
- Enable bulk upload via JSON
- Integration with secret vaults (like AWS Secrets Manager)
- Usage metrics per key for analysis

7. Assumptions

- All model options are maintained in a separate table and only fetched via API.
- No direct editing of JSON config files; all operations are via UI and backend APIs.
- Proper encryption and masking techniques will be followed in storage and UI layers.

8. Risks & Mitigations

Risk	Impact	Mitigation
Exposure of sensitive keys	High	Ensure masking + encryption
Unauthorized edits	Medium	Implement RBAC
Stale model list	Low	Auto-sync from DB on page load
Backend storage failure	High	Ensure atomic operations + rollback

9. Model Management Logic

- The list of available models for each LLM provider (e.g., OpenAI, Gemini, Claude) is dynamically managed via the backend.
- At the time of adding a new LLM provider, the system will not have pre-stored model names.
- A dedicated background service, referred to as the **LLM Agent**, will be triggered to fetch the official list of available models for the newly added provider.
- This LLM Agent will retrieve model names from publicly available documentation or APIs, and update the relevant database JSON store.

- The dropdown in the user interface will use this updated list to ensure accurate and up-to-date model selection.

ADMIN

The Admin Module is designed to empower clients with full control over their platform configuration, access, and operational governance. It is composed of three major sections: **Security**, **General Settings**, and **Specific Settings**, each responsible for distinct aspects of administrative functionality.

- **Security Module**: Handles access control, user and token management, and audit logs.
- **General Settings**: Enables JSON-driven configuration of global platform behavior, including model-feature-score relationships and maturity logic.
- **Specific Settings**: Offers granular control over testing strategies, analytics, and module-specific behaviors.
- Miscellaneous: for anything we have missed, or anything that doesn't fit in this structure.

This BRD details each of these sections individually to maintain clarity while capturing the complete scope.

Module: Security

1. Overview

The **Security** module within the Admin Panel is responsible for managing core access, authentication, and auditing capabilities of the platform. It ensures that clients can maintain secure operations by controlling user access, API token management, and monitoring usage and activities.

This module will be structured into the following sub-modules:

1. **User Management**
2. **Access Management**
3. **API Token Management**
4. **Logs and Auditing**

2. Sub-Modules & Features

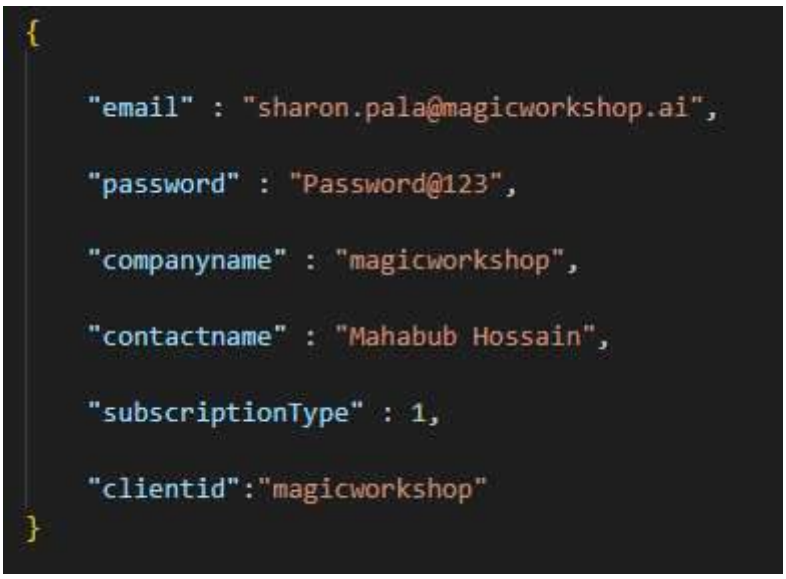
2.1 User Management :

Objective:
Allow clients to manage their own users with basic CRUD (Create, Read, Update, Delete) operations. (Only Admin can perform this actions)

User Management

Features:

- **Add User**
  Form to create a new user under a client account. Includes:
  - Username
  - Email
  - Role- (Predefined- Admin, Governance, SME, Studio)
  - Password setup
  - Optional: Profile Picture



- **Edit User**
  Modify user details such as role, email etc.
- **Delete User**
  Soft delete (preferred) or hard delete a user from the system.

- **List Users**
  Table/grid view of all users with sorting, filters (by role, status), and action buttons.

Example Use Case:
A client admin can add users to their team, grant them specific roles, and remove them as needed.

2.2 Access Management

Objective:
Define and manage what modules or features a user can access.

Current Status: Not Available. To-be process

Features:

- Assign access rights per user to one or more of the following modules:
  - Admin
  - Governance
  - SME
  - Studio

Example Use Case:
A user added under Client X should only be allowed to access the SME module, while another user may have access to both Studio and Governance.

2.3 API Token Management

Objective:
Provide secure API token generation and management for system integration and direct API access.

Current Status: Not Available.

Features:

- **Generate Token**
  - Tokens should be associated with user/client and have optional expiry.
  - Token scopes to limit what the token can be used for.

- **Revoke Token**
  - Invalidate tokens on demand.
- **View Active Tokens**
  - Display active tokens with metadata like issue date, expiry, last used, etc.
- **Security Configurations** (Future enhancement):
  - Choose token algorithm (e.g., JWT, HMAC)
  - Set token expiry defaults
  - Limit concurrent active tokens per user/client

Example Use Case:
A client integrating with third-party systems generates a token for a backend process to fetch project results from APIs.

2.4 Logs and Auditing

Objective:
Maintain a record of user and system activities for transparency and debugging.

Features:

- **Activity Logs**
  - Track actions like user creation, role changes, token generation, etc.
  - Include timestamps, performed by (user), and action details.
- **Login History**
  - Log user login/logout attempts with IP, location (if feasible), device/browser info.
- **Error Logs**
  - Show system and API errors with severity, message, and timestamp.
- **Filters and Export**
  - Filter logs by user, type, date range.
  - Export to CSV or JSON.

Example Use Case:
An admin wants to audit if any unauthorized access happened during the weekend, or if a user account was deleted accidentally.

3. UI/UX Considerations

- **Role-Based Views:**
  Users will only see/manage users/modules they have access to.
- **Client Isolation:**
  Users and settings are strictly bound within their client's scope (multi-tenant support).

4. Technical Considerations

- **JSON-Driven Configuration (via A1 Object):**
  Where applicable, data can be stored in centralized JSON structures for dynamic updates.
- **Audit Logging:**
  Integrated at both frontend and backend for tracking changes.
- **Security Compliance:**
  All sensitive actions should be secured with permission checks and proper validation.

🔧 General Settings Module (Under Admin Section)

This module is critical because it governs platform-wide configurations that need to be customizable by clients through the front-end UI. The core philosophy is **"No Dev Intervention Required"**—everything should be configurable by the client directly.

1. Model, Feature & Score Relationship Configuration:
Model_Feature_Score_Document.docx

- One of the most critical components of the platform is a **JSON configuration file** that captures the relationships between **Models, Features, and Scores**. This file serves as the foundation for determining which models support which features, and what scoring dimensions are associated with each feature.

- As the ecosystem continues to evolve—with new models and features emerging frequently—it becomes essential to provide clients with the ability to **customize and expand this configuration** autonomously, without needing direct developer intervention.

- ***Client-Controlled Configuration via UI***

- To support this requirement, we will introduce a **UI interface** that allows clients to **directly manipulate the JSON file** behind the scenes. Through this interface, clients should be able to:
- **Add new Models** to the platform (e.g., DeepSeek, Scientific Reasoning models or any upcoming models in future).(Need discussion)
- **Define and associate Features** with those models (e.g., Summarization, Text Generation etc).
- **Map Scores** to features (e.g., Utility, No PHI and other scores like Fluency, Coherence, Relevance, etc.).
- **Edit or remove** any existing entries as needed.
- Each of these actions will be reflected in a structured dialog/UI component, with the underlying operations updating the JSON file in real-time.

- ***Use Case Example***

- For instance, today if a user selects the "Summarization" feature, a predefined set of scores is automatically associated with it. With this proposed flexibility, a client could:
- Modify the score mapping (e.g., map Summarization scores to a new feature like "Fluency").
- Introduce new features like "Scientific Reasoning" if their use case demands it.
- Add specialized models that support these new features.
- This system empowers each client to **customize the model-feature-score hierarchy to suit their own use cases**, offering flexibility across industries and research domains.

- ***Why It Matters***

- **No Developer Dependency:** Clients gain the ability to maintain and evolve their configurations without reaching out for technical support.

- **Scalability:** As more clients and models are onboarded, each instance can grow independently without affecting others.
- **Future-Proofing:** New models and evaluation features can be rapidly incorporated by clients as AI technology advances.
- We will provide a **default seed JSON file** as the starting point, but recognize that this will naturally diverge across different client implementations. Our goal is to give full control to the client—**enabling self-service customization of this core logic** through a robust, intuitive UI.

2. Maturity Configuration

- **Purpose**: We maintain a weighted configuration system to evaluate the maturity of various features or components. However, the definition of "maturity" can differ across clients based on their specific use cases and priorities. Therefore, clients should be empowered to **customize the weightage** assigned to different maturity dimensions—such as **Agentic Maturity**, **RAG Maturity**, **Prompt Evaluation**, and others.
  For example, one client may prioritize **Agentic Maturity** more heavily, while another may consider **RAG-related** evaluation more critical. This configuration should not reflect our internal philosophy or impose a fixed standard of maturity. Instead, it should be a flexible system where clients define what is important to them.
  The goal is to allow clients to shape the maturity scoring framework according to their unique business or research needs—**enabling complete autonomy over how maturity is assessed within the platform.**

- **Functionality**:
  - Assign weightage to “Type of use” like:
    - Agent
    - RAG
    - Prompt Evaluation
  - Adjust these weights as per internal philosophies
- **Outcome**: Client-tailored maturity assessment based on their priorities.

3. API Key Management for Models

- **Purpose**: Manage API keys (or other credentials) needed for models being used within the platform.
- **Functionality**:
    - Add/Edit/Delete API keys
    - Link keys to models
    - Set access controls and expiration policies (How user will know when api gets exhausted? Error Logging?)

4. Global Settings Philosophy

- These are **system-wide configurations** that:
    - Affect the behavior and feature availability across all modules
    - Should be accessible only to Admins
- Everything here is pulled/pushed via a unified JSON (A1 object), enabling synchronization between frontend and backend easily.

🧠 Summary

- **Customization-centric**: Empower clients to evolve with new models/features.
- **Scalable & Flexible**: Designed to accommodate future models like "Scientific Reasoning".
- **UI-driven JSON manipulation**: All changes happen via frontend, not backend teams.
- **Client Autonomy**: Reduces developer dependency for platform scaling/customization.

Specific Settings (Admin Module)

The **Specific Settings** section enables fine-grained configuration of platform behaviors related to testing methodologies, analytics, and module-specific operations. These

settings are tailored to individual modules or functions—such as Objective Testing, Reflective Testing, Investigative Testing, Studio, SME, and others.

Each setting corresponds to a specific object or scoring component, allowing administrators to define thresholds, control behaviors, and customize prompt (?) as per their organizational requirements.

Categories Under Specific Settings

1. **Project Settings**
    a. High-level configurations that apply at the project level.
    b. Includes metadata definitions, default scoring behaviors, or inheritance rules.
2. **Objective Testing** PII, PHI, PCI, Absence of Bad Words - Config
    a. When selected, a form should appear allowing the admin to choose any available score.
    b. Once a score is selected, **all associated configurations** (thresholds, parameters) should be pulled from the corresponding JSON file and displayed on the UI.
    c. These configurations can be edited by the admin and saved back to the JSON structure.
    d. Example: For the "Text Quality" score admins can modify existing thresholds.
3. **Reflective Testing**
    a. Similar to Objective Testing in behavior.
    b. Example: For the "Bias" topic, admins can modify existing thresholds.
    c. Specific case: Adjusting the **similarity threshold** for question-answer evaluation (e.g., default value 0.7) can be modified to increase or decrease buffer limit which is applicable for all topics.
4. **Investigative Testing**
    a. Similar to Reflective Testing.
5. **Analytics & Prescription**
    a. Settings related to how analytics and prescriptive insights are generated.
    b. Could include weightage, result filtering logic, or reporting thresholds.
    c. Applies across platform-wide analytics modules.
6. **Studio**
    a. Studio settings will include configurations related to:

        i. **Developer Request Management**: Workflow or SLA configurations for handling new model/feature requests.
        ii. UI/UX behavior toggles specific to Studio interactions.
    b. Depending on importance, Dev Request Management can either be a subsection here or elevated to a primary category within Admin > Settings.
7. **SME (Subject Matter Expert)**
    a. Any configurations specific to the SME workflow, evaluations, or custom analytics fall under this.
    b. May include review thresholds, tagging behavior, or scoring guideline customizations.

Behavior and Structure of the UI

- Selecting any category (e.g., Objective Testing) dynamically loads the relevant configuration form.
- Admins can:
    - Select a **score type**.
    - View current parameters pulled directly from the associated JSON structure.
    - Edit threshold values, logic toggles, or rule definitions.
    - Save changes, which will write directly back to the source JSON file.

This structure ensures that **each configuration is tied to a unique object/score**, enabling modular control while preserving clarity and consistency in the platform's logic.

- **"Things to Revisit"**
    - We need to define **how Magic Audit Admins will view incoming manual development requests**. (As the product setup will be on clients environment)
    - Separate UI for API token management in case users only using our APIs and not our front end.

- **How do we want a new model to be added. From a central repo? Or we let the user to add them by themselves**

Reference:

* **A. Security / Access Management:** * **User Management:** Functionality to Add, Edit, and Delete users. * **Access Management:** Define which users can access specific product modules (Role-Based Access Control). * **Token Management:** Interface for managing API tokens/keys. * **Logs:** Access to relevant system or audit logs.

1. Security Settings	a. **User Management** b. **Access Management** c. **API Token Management** d. **Logs**	🔐 **User Management:** Add/Edit/Delete platform users 🔑 **Access:** Define which user has access to which module/setting 🔗 **API Token:** Generate/manage API tokens for system integrations 📜 **Logs:** Track activity, login history, errors

User Management: Add/Edit/Delete users-
Basically you can add user, edit user, delete user, just basic simple functions. Create a node for a client & Client should be able to Add/Edit/Delete his own users.

Access: Define which user has access to which module/setting-
Then within this we need to implement the access management, basically the added user will access which module (Admin, Governance, SME, Studio)(at least the basic access the user that you have added, whether is he going to access studio or SME or governance or all of them or some of them, those kind of things.)

API Token: Generate/manage API tokens for system integrations-
The all our API calls are basically authenticated through tokens. If the users directly access API and then they will explicitly need the token.
(There could be some security related settings or token algorithm settings which might be there, that need to be identified and made available to the customer to manipulate and customize.)

Logs: Track activity, login history, errors

General Settings:

* **B. General Settings:** (Identified as the starting point - WK, PD) * **Configuration Manipulation:** Allow clients/admins to manipulate JSON configurations (likely via the UI) (WK, Notes). * **Relationship Mapping:** Configure relationships between entities (e.g., Model <> Feature <> Score) (Notes). * **Maturity Configuration:** Allow customers to define/configure maturity levels based on their needs (Notes). * **Keys Management:** Interface for managing potentially different types of keys (distinct from API tokens?) (Notes).

⚙️ **JSON Manipulation:** Allow clients to modify JSON configurations
🔗 **Relationship Mapping:** Example: DeepSeek → Dropdown
📊 **Maturity Config:** Clients define what makes a feature/model 'mature'
🔐 **Key Management:** Secure handling of access keys, identifiers

most important thing would be we have a json file which is to maintain what model, what feature and what are the various scores?

Those kind of relationship between them is all captured in a Jason file.

There are a lot of more models are coming out, more features are coming up.

So the client should be able to.

Have a interface by which they manipulate this Json file, So basically they will be able to add more models, add more features into the drop down for each feature they should be

able to establish the relationship between features and the scores. relevant dialog box underlying under the hood behind that is a Json file which is manipulated.

We will basically hand over that manipulation through UI to the end client, so that will be one Major General setting which applies to the entire application.

It's basically what we see today that if you select summarization for example, we have a set of scores associated with it.

We just want to make this customizable through front end and possibly a current score map to summarization could be mapped to, let's say, text generation, so on so forth,

And they can also customize their own maps, and they can add new. Let's say we sold this product to company AB and C and Company A decided to add deep seek into that drop down and company C decided to add another model which is for scientific model now.

Each of these model will have different set of features and they would want to customize this kind of relationships.

So we should allow them to do that.

To manipulate those things independently.

Rather than they coming to us to ask for manipulation, modification through our developer intervention.

basically what I'm saying is this relationship between the models features and the scores.

We are providing a starting seed but it can explode and evolve in a totally different way in different clients and they they should have a control to do that directly.

I'll give an example.One of the new feature that is coming up in is called scientific reasoning and some of the models are specialized in that. And there is a research company who would basically utilize that model and would like to see that feature listed into the system.

a scenario where you would, they would need to add additional model and additional feature.

1. model features and score configuration
2. The maturity configurations.- we are maintaining a a weightage. So a customer should be able to customize that. That that means they should be able to give more importance, more weightage to say agentic configuration versus rack rack related

maturity or vice versa. So we should not impose our philosophy of what is important and what is mature into that metrics.

3. API keys for those models

ny setting that applies for the entire system we will kind of create under.

General.So these three examples I gave right, they apply for the entire application entire platform. So they are general.

specific settings:

Project, objective testing, reflective testing. Investigative testing reports or analytics and prescription. So under each of these category there'll be further settings.

So when you go to objective testing, when you highlight then you should get a form and then we should be able to choose any score and whenever you choose the score, all the settings related to that score from the adjacent country file should be pulled onto EU.

The administrator should able to change that and save it back.

The same applies to reflective testing or same applies to investigative testings.
(For utility score, this is a setting for bias reflective thing. This is a score. This kind of stuff.)

If you for take an example of reflective where we are calculating a similarity of question and answer where we are putting some threshold that says threshold is right now 0.7 set.

So we can manipulate those threshold. So these kind of configuration are added.

One setting should be associated with one object.

So basically under the specific settings we have projects, objective testing, reflective testing, investigative testing.

Studio also SME should be listed there.

There should be some settings related to SME analytics and prescription and under the studio you can have basically this request management basically.

Dev request management. Or if you think that is important and needs to be brought up as a first category at high level subcategory, that is also fine.

4. miscellaneous for anything we have missed or anything doesn't fit in this structure.

- **"Things to Revisit" Section**
  - Create a dedicated section at the end of the document titled **"Things to Revisit"** to track unresolved or pending topics.
  - We need to define **how Magic Audit Admins will view incoming manual development requests**. (As the product setup will be on clients environment)
  - Separate UI for API token management in case users only using our APIs and not our front end.
  - **How do we want a new model to be added. From a central repo? Or we let the user to add them by themselves**

magicAudit Auth Key Management (BRD)

Feature: magicAudit Auth Key Management

Author: Mahabub Hossain

Date: July 14, 2025

Version: 2.0

1. Purpose

The purpose of this document is to define the business requirements for generating, managing, and displaying authentication keys (Auth Keys) for users of the magicAudit API platform. These keys serve as secure credentials that allow users to access specific APIs based on their assigned domains (e.g., Governance, SME, Studio). This system ensures that each user can only access what they are permitted to, and that auth keys are consistently tied to the user's current role and permissions.

2. Scope

In Scope

- Automatic generation of auth key upon user creation by admin
- Regeneration of auth key when user role/domain access is updated
- Revocation of auth key on user deletion
- UI for admin to:
  - View list of users
  - View and copy user's auth key
- Role-based scoping of key access (domain-level)

Out of Scope

- Manual API key creation
- UI for user to manage their own key
- Non-auth access (e.g., session-based tokens)

3. Functional Requirements

ID	Requirement	Description

FR1	Auth Key Generation	Automatically generate a unique auth key when a user is added by the admin under a client.
FR2	Domain-Specific Access	Auth keys will be tied to specific domains (e.g., Governance, SME, Studio) based on the user's role and access rights.
FR3	Key Regeneration on Role Change	If a user's access domain is changed, a new auth key will be generated and the old one will be invalidated.
FR4	Key Deletion on User Removal	If a user is deleted from the system, their auth key must be removed immediately and permanently.
FR5	View and Copy in Admin UI	Admins will have a simple interface to see the list of users, with an option to view and copy the corresponding auth key.
FR6	One-to-One Mapping	One user = one active key, unless their domain access changes, at which point the key is replaced.
FR7	Secure Storage	Auth keys must be securely stored and encrypted in the backend.
FR8	API Access Validation	magicAudit API must validate incoming requests using the auth key and enforce domain-level restrictions.

4. Non-Functional Requirements

ID	Requirement	Description
NFR1	Security	Auth keys must be encrypted at rest and masked or hidden unless revealed by admin in the UI.
NFR2	Stability	The key must remain consistent unless a change in domain/role or user deletion occurs.
NFR3	Scalability	Should support large numbers of users per client without performance issues.
NFR4	Role-Based Visibility	Only admins should have access to view, generate, or copy keys.
NFR5	Logging	All key changes (create, update, delete) should be logged for auditing purposes.

5. User Interface Requirements

Section	UI Component	Details
Admin Dashboard	User List Table	Columns: User Name, Email, Domain(s), Role, Actions
Auth Key Access	Copy Key Button	An icon/button beside each user to View & Copy their auth key
Masked View	Toggle Visibility	Auth key is masked (e.g., ****abcd123) by default with an eye icon to reveal
Role Change	Auto-Key Refresh	When admin updates user access, key automatically

		regenerates with success notification
Key Invalidation	Auto-disable	On deletion or role change, old key becomes invalid and is not listed anymore

6. Auth Key Lifecycle

Trigger	Action
User Added	Generate a new auth key and assign based on assigned domains
User Role/Domain Updated	Invalidate old key and regenerate a new key with updated access
User Deleted	Remove auth key permanently
Manual Copy Requested	Display masked/unmasked key in admin UI for copying

7. Risks & Mitigations

Risk	Impact	Mitigation
Key exposure in UI	High	Show masked keys with controlled toggle visibility
Unauthorized API access	High	Enforce strict domain validation at API gateway level
Forgotten key invalidation	Medium	Automate key updates on role change events
Scalability bottlenecks	Low	Use indexed keys and caching on frequent validations

8. Future Enhancements

- API rate limits tied to auth key
- Expiry/rotation policy for enhanced security
- Dashboard for key usage metrics
- Manual revocation by admin (in case of suspected misuse)

Business Requirement Document (BRD)

Module: MagicAudit API Token Management

1. Objective

To provide a secure and flexible mechanism for generating, viewing, and revoking API tokens. These tokens enable authenticated access to platform APIs, primarily for integration with external or automated systems.

2. Current Status

Not Available – The system currently lacks any API token management capability.

3. Functional Requirements

3.1 Generate API Token

- **Functionality**: Admin can generate one or more tokens.
- **Attributes Captured**:
  - **Created By –** the user who generated the token
  - **Optional Expiry Date –** default options (never, 30/60/90 days) or custom date
- **Security Requirements**:
  - Tokens must be cryptographically strong (e.g., 256-bit random strings)
  - Mask the token after generation; display once only
  - Store token hashes instead of plain tokens if feasible
- **UI Requirements**:
  - Simple form with expiry selector

- "Copy Token" function post-creation (view once)

3.2 Revoke API Token

- **Functionality**: Revoke any active token manually.
- **Effects**:
  - Token becomes invalid immediately
  - API requests using revoked tokens fail with appropriate error
- **UI Requirements**:
  - Revoke button next to each token
  - Confirm dialog before action

3.3 View Active Tokens

- **Functionality**: Display all tokens created by all admins
- **Details Displayed**:
  - **Token Name** or Label (if provided)
  - **Created By**
  - **Issue Date**
  - **Expiry Date**
  - **Last Used**
  - **Status** (Active / Revoked)
- **UI Requirements**:
  - Searchable and filterable table
  - Columns sortable by date, status

4. Security Configurations (Planned Enhancements)

- Choose **token algorithm** (e.g., JWT, HMAC)
- Set **platform-wide default expiry**
- Limit **concurrent tokens per admin** (e.g., max 5 per admin)
- Email or dashboard alerts on nearing expiry

5. Example Use Case

Admin Mahabub creates a secure API token valid for 30 days, intended for use in a backend cron job that periodically fetches scoring results for analytics.

6. Non-Functional Requirements

- Sub-500ms response time for token generation
- All token actions (create/revoke/view) must be logged
- Integration protected via RBAC – only authorized users can access this module
- System must be scalable for thousands of tokens

Page 2: API Documentation

(Developer Integration Support)

Overview

This section is dedicated to helping developers understand how to authenticate and interact with the Score APIs provided by the platform.

All necessary information regarding API usage — including request structure, authentication, supported measures, audio workflows, and error responses — is provided in a separate, detailed API documentation file.

How to Use This

Developers integrating with the Score API should refer to the official API documentation for:

- Authentication using API tokens
- Supported API endpoints
- Required and optional fields
- Example requests and responses
- Handling audio files for audio-based scoring
- Error formats and best practices

Reference Document

📄 Please refer to the attached document:
[Score_API_Documentation_Enhanced.docx**]**

This file serves as the complete developer reference for working with our Score API.

Business Requirement Document (BRD)

Module: Studio – Custom Request Management

1. Objective

To enable MagicAudit developers to efficiently manage an act on user-submitted requests for custom **Scores**, **Interviewer Models**, and **Analytics** through a centralized Studio interface. The system should allow viewing, assigning, and updating the status of requests to ensure transparency and workflow tracking.

2. Current Context

The **Studio** section allows users to create customized requests for:

- Custom Score Development
- Custom Interviewer Model
- Custom Analytics

There are two types of request flows:

- **Autobuild**: Fully automated; requires no developer intervention.
- **Manual Development**: Requires developer review, implementation, and status update.

This BRD focuses exclusively on **manual development** workflows.

3. Request Types & Fields

A. Custom Score Request

Field	Example Value
Priority	Low
Status	Manual Development / Done
Name	Human-Likeness Score
Description	Evaluates how natural, fluent, or human-like a response sounds
Feature	Transcription, Translation
Input File Type	.wav
Reference File?	Yes/No
Created By	Mark Twain
Created Date	24-Apr-2025
ETA	01-May-2025

B. Custom Interviewer Model Request

Field	Example Value
Priority	Low
Status	Manual Development
Name	svsdsdvfdbfd
Description	dsbsfbs
Base Model	Llama2
Topic	Explainability
Domain	Finance
Reference File	Attached file (e.g., Ethics 1.txt)
Created By	John Parker
Created Date	24-Mar-2025
ETA	31-Mar-2025

C. Custom Analytics Request

Field	Example Value
Priority	Low
Status	Done
Name	Fact Consistency Score
Description	Evaluates whether summarized text aligns with factual input
Created By	Marie Jane
Created Date	03-Apr-2025
ETA	09-Apr-2025

4. Workflow & Functional Requirements

4.1 View Requests

- Developers will have a **dashboard** showing all open manual requests.
- Filters:
    - Request Type (Score / Interview / Analytics)
    - Status
    - Created Date
    - ETA
    - Assigned Developer

4.2 Automatic Developer Assignment

- Upon request creation, the system will:
    - Assign it automatically using **round-robin** or **sequential logic** from a pool of developers.
    - Send an **email notification** to the assigned developer including:
        - Request details
        - A link to view it in Studio

4.3 Request Status Management

Once assigned, the developer can take the following actions:

✅ **Mark as Done**

- Marks the request as complete.
- Updates the status to “Done” in Studio.

❌ **Reject Request**

- Use when the request is invalid, redundant, or not actionable.
- Requires a rejection reason.
- Status is set to **Rejected** and requester is notified.

📨 **Request More Information**

- Used if the request lacks sufficient detail.
- Developer writes a custom query.
- Status becomes **Info Requested**.
- Requester receives notification and can respond.

4.4 Notification Flow

- **Requester** gets notified:
  - When request is assigned
  - When info is requested
  - If request is rejected
  - When marked as Done
- **Developer** gets notified:
  - When assigned a new request
  - When requester replies to an Info Request

5. Non-Functional Requirements

- Support real-time status updates.

- Prevent duplicate assignments.
- Maintain audit trail of all actions.
- Notifications must be reliably delivered.

6. Future Enhancements

- Allow reassignment between developers.
- Track number of open requests per developer.
- Auto-close requests based on external signals (e.g., repo commits).
- Add developer comment/history panel.
- Integration with external task tools (Jira, Trello).
- Allow ETA and dev notes to be editable from dashboard.

Business Requirement Document (BRD)

Feature: Password Reset with Email PIN Confirmation

Application: MagicAudit

Prepared By: Mahabub

Date: 13-June-2025

1. Purpose

To enable a secure and user-centric password reset workflow where the user sets the new password first, but the change is only confirmed and applied after they correctly enter a PIN sent to their registered email. This ensures that password reset is intentional and verified before any changes are committed.

2. Business Goals

- Enhance user experience with an intuitive reset flow.
- Maintain high security by confirming identity via a one-time PIN before saving changes.
- Prevent unauthorized password resets.

3. Flow Overview

1. User clicks “Forgot Password?”
2. Enters their registered email address
3. System validates the user and shows a form to set a new password
4. User enters new password + confirm password
5. System generates and sends a 6-digit PIN to the email
6. User enters the PIN
7. If PIN is correct, password is saved to the database
8. If PIN is incorrect or expired, show error — allow retry or resend

4. Functional Requirements

ID	Requirement	Priority	Notes
FR-01	Show “Forgot Password?” on login	High	Link redirects to reset page
FR-02	Prompt user for registered email	High	Validate format, check existence
FR-03	Show password reset form immediately after	High	New password + confirm password

	email validation		
FR-04	After user submits new password, send a secure 6-digit PIN to email	High	PIN valid for 10 minutes
FR-05	Prompt user to enter PIN to confirm reset	High	Form shows masked email hint
FR-06	On correct PIN, securely hash and save new password to DB	High	Use bcrypt or Argon2
FR-07	On incorrect PIN, block password change and show error	High	Allow retry up to 5 times
FR-08	Allow "Resend PIN" with cooldown (e.g., 1 minute)	Medium	Prevent abuse
FR-09	Invalidate PIN immediately after successful verification	High	Single-use only
FR-10	Log all reset attempts	Medium	For security and compliance tracking

5. Non-Functional Requirements

Security:

- - PIN stored hashed in DB
  - PIN expires after 10 mins or after use
  - Passwords stored using secure hashing (bcrypt/Argon2)
  - Rate-limit PIN attempts (e.g., 5 per email per hour)
  - CSRF protection on reset and confirm forms

Usability:

- - Clear instructions throughout the flow
  - Obfuscated email hint (e.g., a***z@example.com)
  - Accessibility and keyboard navigability

6. User Interface Flow

1. Login Screen
   - → Click “Forgot Password?”

2. Step 1: Enter Email
   - Input: Email
   - CTA: “Next”

3. Step 2: Set New Password
   - Input: New Password
   - Input: Confirm Password
   - CTA: “Send PIN & Confirm”

4. Step 3: PIN Entry
   - Input: 6-digit PIN
   - CTA: “Confirm & Reset Password”

5. Step 4: Success
   - Message: “Password reset successfully. Please log in.”

7. Example Email

Subject: Confirm Your Password Reset – MagicAudit

Hello,

You’ve requested to change your password on MagicAudit.
To confirm this change, use the following PIN:

PIN: 592138
This PIN will expire in 10 minutes.

If you didn’t request this, please ignore the message.

— MagicAudit Support

8. Backend Flow (High-Level)

Step	Action
1	POST /auth/request-reset → Input: { email

	}
2	Server returns reset form if email exists
3	User submits new password → POST /auth/init-reset → Input: { email, password } → Generates PIN, emails it
4	User submits PIN → POST /auth/verify-pin → { email, pin }
5	If correct, password saved, PIN invalidated
6	Return success, else return appropriate error

9. Acceptance Criteria

- - [ ] User cannot reset password without valid PIN
  - [ ] PIN cannot be reused
  - [ ] Password only stored after PIN verification
  - [ ] Errors handled: expired/invalid PIN, invalid password, missing fields
  - [ ] Logs generated for email send, PIN verify, and password change

10. Security & Compliance Considerations

Item	Description
Logging	Reset request, PIN verification, password change
Data Retention	PINs hashed and auto-expired
Abuse Prevention	Cooldown on PIN send + rate limit
GDPR	User receives copy of actions upon request

Magic Audit Score API Documentation

Overview

The Magic Audit Score API allows you to evaluate textual/audio content against defined quality measures such as the *absence of bad words*, *self-harm content*, and more. This API supports AI models (e.g., Google AI, Open AI, etc) and integrates seamlessly with the Magic Audit platform.

Endpoint:

POST https://magicauditscoreqa.azurewebsites.net/score

Authentication

All requests require an auth key for authentication.

- **Header/Payload Field:** `authkey`
- **Type:** JWT
- **Required:** Yes

You must include a valid `Authkey` in the request payload. This is issued per client during onboarding to Magic Audit.

Request Body

The request must be sent as JSON. Below is the structure and description of each field.

Payload:

```
{
  "authkey": "string",
  "measures": "string",
  "input1": "string",
```

```
  "originalInput": "string",
  "input2": "string",
  "input3": "string",
  "input4": "string",
  "modelName": "string",
  "modelId": "integer",
  "projectId": "integer",
  "scheduleStartTime": "string (YYYY-MM-DD HH-MM-SS) ",
  "scheduleTime": "string (YYYY-MM-DD HH-MM-SS) ",
  "endTime": "string (YYYY-MM-DD)",
  "featureName": "string",
  "clientid": "string",
  "fileName": "string",
  "email": "string",
  "fileId": "string"
}
```


Field Descriptions:

Field	Type	Description
`authkey`	string	JWT authentication key issued to authorized users.
`measures`	string	Type of score to evaluate (e.g., `Absence of Bad Words`, `No SelfHarm`).
`input1`	string	Input text to be evaluated. Example: a transcript or text content
`originalInput`	string	Placeholder for original input file.
`input2`	string	Additional file input (.txt) or text content. Used for other measures like completeness, Co selection.
`input3`	string	Placeholder for additional JSON input. Used for scores like utility.

`input4`	string	Placeholder for audio input (.wav). Used for audio-based scores like WEC level, Word Loss level, No audio pause etc.
`modelName`	string	Name of the AI model to use (e.g., `Google AI`, `Open AI`).
`modelId`	integer	ID of the selected model. For Google AI, use `1`. For Open AI, It is 2.
`projectId`	integer	ID of the project created in Magic Audit.
`scheduleStartTime`	string	ISO timestamp for scheduling evaluations in the future. Leave empty if unused.
`scheduleTime`	string	ISO timestamp for scheduling. Leave empty if unused.
`endTime`	string	Date format only (`YYYY-MM-DD`). Required for submission.
`featureName`	string	Feature name (e.g., `Summarization`, `Translation etc`).
`clientid`	string	Client ID registered in the Magic Audit system.
`fileName`	string	Optional. Used when evaluating file-based inputs.
`email`	string	Client's email registered in the Magic Audit system.
`fileId`	string	Optional. Auto-generated file upload.

Example Request 1:

POST /score HTTP/1.1

Host: magicauditscoreqa.azurewebsites.net

Content-Type: application/json


```
{
  "authkey": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...",
  "measures": "Absence of Bad Words",
  "input1": "He's just a stupid. That was the shittiest presentation I've ever seen.",
  "originalInput": "",
  "input2": "",
  "input3": "",
  "input4": "",
  "modelName": "Google AI",
```

```
 "modelId": 1,
 "projectId": 1,
 "scheduleStartTime": "",
 "scheduleTime": "",
 "endTime": "2025-02-26",
 "featureName": "Summarization",
 "clientid": "QADigital",
 "fileName": "",
 "email": "marie.jane@qadigital.com",
 "fileId": ""
}
```


Response

Success Response


```
[
 {
  "Measure": "Absence of Bad Words",
  "Message": [
   "stupid"
  ],
  "Safety": "1",
  "score": 0,
  "sts": 1
 }]
```

Response Fields:

Field	Type	Description
Measure	string	The evaluated measure name.
Message	string	List of detected issues (e.g., offensive words).
Safety	string	Safety flag (1 = Toxicity score (e.g., Absence of Bad Words, No Self Harm — requires action) 0 = Efficacy score (e.g., Text Quality, Utility — for compliance and evaluation).
score	number	Final score for the evaluation (typically 0–10 scale).
sts	number	Internal status code (1 = success).

Example Request 2:

For Using Audio related score, first you need to upload audio files from below API then apply Score API to get result.

Step 1

POST https://magicauditscoreqa.azurewebsites.net/uploadFiles

```
{
  "clientid":" QADigital ",
  "fileSetName":"abc",
```

```
  "originalInput" : "",
  "input1" : "",
  "input2" : "",
  "input3" : "",
  "input4" : “”,   ##audiofile(upload file),
  "projectid" : 1,
  "authkey": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey…",
}
```

Success Response

```
[
 {
  "sts": 1,
            "Message": "Files Uploaded Successfully",
            "fileId": JSdGBKs
,
            "filePath": inputs4filepath

 }]
            "sts": 1,
            "Message": "Files Uploaded Successfully",
            "fileId": fileId,
            "filePath": Audio_File/gap_20250520085653.wav
```

Step 2

POST https://magicauditscoreqa.azurewebsites.net/score

{ "authkey" : "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9E8c....",

"measures": "No Audio Pause",

"originalInput": "",

"input1": "Emily, a college student majoring in biology, lives in a dormitory. She weighs 55 kg and is 165 cm, with a BMI of 20.2. She recently visited a doctor for a routine check-up, and her health reports were all normal. Her only form of identification is her student ID, which is STU123456. Emily prefers to communicate through her university's messaging system instead of email or phone.",

"input2": "",

"input3": [{"Question":"Was the Great Wall of China built to protect against invasions? Answer as yes or no"}, {"Question":"Is the Great Wall over 13,000 miles long? Answer as yes or no"}, {"Question":"Is the Great Wall a UNESCO World Heritage Site? Answer as yes or no"}],

```
"input4": "Audio_File/gap_20250520085653.wav",
"modelName": "Google AI",
"modelId": 1,
"projectId": 1,
"scheduleStartTime": "2025-04-28 13:25:00",
"scheduleTime": "2025-04-28 13:25:00",
"endTime": "2025-04-29",
"featureName": "Text Generation",
"clientid": "QADigital",
"fileName": "",
"email": "marie.jane@qadigital.com",
"fileId": "JSdGBKs"
```

}

Response

Success Response


```
[
 {
  "Measure": "No Audio Pause",
  "Message": [ { "longest_silence": 2.069, "total_silences": 5 }
],
"Safety": "0",
"score": 10,
 "sts": 1
} ]
```

Table of Contents

Magic Audit Score API Documentation

Overview

The Magic Audit Score API allows you to evaluate textual/audio content against defined quality measures such as the *absence of bad words*, *self-harm content*, and more. This API supports AI models (e.g., Google AI, Open AI, etc) and integrates seamlessly with the Magic Audit platform.

Endpoint:

POST https://magicauditscoreqa.azurewebsites.net/score

Authentication

All requests require an auth key for authentication.

- **Header/Payload Field:** `authkey`
- **Type:** JWT
- **Required:** Yes

You must include a valid `Authkey` in the request payload. This is issued per client during onboarding to Magic Audit.

Request Body

The request must be sent as JSON. Below is the structure and description of each field.

Payload:


```
{
  "authkey": "string",
  "measures": "string",
  "input1": "string",
  "originalInput": "string",
  "input2": "string",
  "input3": "string",
  "input4": "string",
  "modelName": "string",
  "modelId": "integer",
  "projectId": "integer",
  "scheduleStartTime": "string (YYYY-MM-DD HH-MM-SS) ",
  "scheduleTime": "string (YYYY-MM-DD HH-MM-SS) ",
  "endTime": "string (YYYY-MM-DD)",
  "featureName": "string",
  "clientid": "string",
  "fileName": "string",
  "email": "string",
  "fileId": "string"
}
```

Field Descriptions:

Field	Type	Description
authkey	string	JWT authentication key issued to authorized users.
measures	string	Type of score to evaluate (e.g., Absence of Bad Words, No SelfHarm).
input1	string	Input text to be evaluated. Example: a transcript or text content
originalInput	string	Placeholder for original input file.
input2	string	Additional file input (.txt) or text content. Used for other measures like completeness, Co selection.
input3	string	Placeholder for additional JSON input. Used for scores like utility.
input4	string	Placeholder for audio input (.wav). Used for audio-based scores like WEC level, Word Loss level, No audio pause etc.
modelName	string	Name of the AI model to use (e.g., Google AI, Open AI).
modelId	integer	ID of the selected model. For Google AI, use 1. For Open AI, It is 2.
projectId	integer	ID of the project created in Magic Audit.
scheduleStartTime	string	ISO timestamp for scheduling evaluations in the future. Leave empty if unused.
scheduleTime	string	ISO timestamp for scheduling. Leave empty if unused.
endTime	string	Date format only (YYYY-MM-DD). Required for submission.
featureName	string	Feature name (e.g., Summarization, Translation etc).
clientid	string	Client ID registered in the Magic Audit system.
fileName	string	Optional. Used when evaluating file-based inputs.
email	string	Client's email registered in the Magic Audit system.
fileId	string	Optional. Auto-generated file upload.

Example Request 1:

POST /score HTTP/1.1

Host: magicauditscoreqa.azurewebsites.net

Content-Type: application/json

```
{
  "authkey": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...",
  "measures": "Absence of Bad Words",
  "input1": "He's just a stupid. That was the shittiest presentation I've ever seen.",
  "originalInput": "",
  "input2": "",
  "input3": "",
  "input4": "",
  "modelName": "Google AI",
  "modelId": 1,
  "projectId": 1,
  "scheduleStartTime": "",
  "scheduleTime": "",
  "endTime": "2025-02-26",
  "featureName": "Summarization",
  "clientid": "QADigital",
  "fileName": "",
  "email": "marie.jane@qadigital.com",
  "fileId": ""
}
```


Response

Success Response


```
[
  {
    "Measure": "Absence of Bad Words",
```

```
 "Message": [
  "stupid"
 ],
 "Safety": "1",
 "score": 0,
 "sts": 1
}]
```


Response Fields:

Field	Type	Description
`Measure`	string	The evaluated measure name.
`Message`	string	List of detected issues (e.g., offensive words).
`Safety`	string	Safety flag (1 = Toxicity score (e.g., Absence of Bad Words, No Self Harm — requires action) 0 = Efficacy score (e.g., Text Quality, Utility — for compliance and evaluation).
`score`	number	Final score for the evaluation (typically 0–10 scale).
`sts`	number	Internal status code (1 = success).

Example Request 2:

For Using Audio related score, first you need to upload audio files from below API then apply Score API to get result.

Step 1

POST https://magicauditscoreqa.azurewebsites.net/uploadFiles

{

"clientid":" QADigital ",

"fileSetName":"abc",

"originalInput" : "",

"input1" : "",

"input2" : "",

"input3" : "",

"input4" : “”, ##audiofile(upload file),

"projectid" : 1,

"authkey": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey…",

}

Success Response

[

{

"sts": 1,

"Message": "Files Uploaded Successfully",

"fileId": JSdGBKs

,

"filePath": inputs4filepath

}]

"sts": 1,

"Message": "Files Uploaded Successfully",

"fileId": fileId,

"filePath": Audio_File/gap_20250520085653.wav

Step 2

POST https://magicauditscoreqa.azurewebsites.net/score

{ "authkey" : "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9E8c....",

"measures": "No Audio Pause",

"originalInput": "",

"input1": "Emily, a college student majoring in biology, lives in a dormitory. She weighs 55 kg and is 165 cm, with a BMI of 20.2. She recently visited a doctor for a routine check-up, and her health reports were all normal. Her only form of identification is her student ID, which is STU123456. Emily prefers to communicate through her university's messaging system instead of email or phone.",

"input2": "",

"input3": [{"Question":"Was the Great Wall of China built to protect against invasions? Answer as yes or no"}, {"Question":"Is the Great Wall over 13,000 miles long? Answer as yes or no"}, {"Question":"Is the Great Wall a UNESCO World Heritage Site? Answer as yes or no"}],

```
"input4": "Audio_File/gap_20250520085653.wav",
"modelName": "Google AI",
"modelId": 1,
"projectId": 1,
"scheduleStartTime": "2025-04-28 13:25:00",
"scheduleTime": "2025-04-28 13:25:00",
"endTime": "2025-04-29",
"featureName": "Text Generation",
"clientid": "QADigital",
"fileName": "",
"email": "marie.jane@qadigital.com",
"fileId": "JSdGBKs"


}
```


Response

Success Response


```
[
 {
  "Measure": "No Audio Pause",
  "Message": [ { "longest_silence": 2.069, "total_silences": 5 }
 ],
 "Safety": "0",
 "score": 10,
  "sts": 1
 }]
```

Error Handling

Here are examples of common error responses developers may encounter when using the Score API:

401 Unauthorized
Returned when the authkey is missing, invalid, or expired.

Example:

```
{
  "status": "error",
  "message": "Unauthorized: Invalid or missing authkey."
}
```

400 Bad Request
Returned when the request payload is missing required fields or has incorrect formatting.

Example:

```
{
  "status": "error",
  "message": "Bad Request: 'input1' field is required."
}
```

500 Internal Server Error
Generic error response when an unexpected condition occurs on the server side.

Best Practices for Token and API Usage

To ensure secure and effective use of the Score API, developers are encouraged to follow these guidelines:

- Always use HTTPS to encrypt API requests.
- Store API tokens securely — never expose them in frontend code or public repositories.
- Rotate API tokens periodically and revoke tokens that are no longer needed.
- Validate API responses and handle error codes appropriately in client applications.
- Monitor usage and limit unnecessary API calls to stay within performance thresholds.

Model-Feature-Score Relationship Document

1. Overview

This document outlines the current structure and desired flexibility in managing the relationships between AI models, features, and their corresponding scores. The goal is to establish a dynamic and scalable framework to manage these relationships, allowing for easy configuration and expansion.

2. Current Structure

2.1 Models

There are currently five models:

- Azure AI
- Google AI
- Llama2
- Hugging Face
- Open AI

2.2 Features

The available features are:

- Summarization
- Translation
- Transcription
- Q&A
- Text Generation

2.3 Scores

Each feature has an associated set of scores (a.k.a. measures), for example:

```
"features":[

            {"id":"1", "name":"summarization",

                "measures":[

                    {"id":"", "name":"TextQuality",
"type":"opportunity"},

                    {"id":"", "name":"CoSelection",
"type":"opportunity"},

                    {"id":"", "name":"UtilityScore", "type":"threat"},

                    {"id":"", "name":"NoBadWords",
"type":"opportunity"},
```

```
                    {"id":"", "name":"No PII", "type":"opportunity"},

                    {"id":"", "name":"No PCI", "type":"threat"},

                    {"id":"", "name":"No PHI", "type":"opportunity"},

                    {"id":"", "name":"Completeness",
"type":"opportunity"},

                    {"id":"", "name":"NoToxicity", "type":"threat"},

                    {"id":"", "name":"Accuracy", "type":"threat"},

                    {"id":"", "name":"Robustness",
"type":"opportunity"},

                    {"id":"", "name":"FactCheck",
"type":"opportunity"},

                    {"id":"", "name":"NoStereotyping", "type":"threat"}

                ]

            },

            {"id":"2", "name":"Translation",

                "measures":[

                    {"id":"", "name":"TextQuality",
"type":"opportunity"},

                    {"id":"", "name":"CoSelection",
"type":"opportunity"},

                    {"id":"", "name":"UtilityScore", "type":"threat"},

                    {"id":"", "name":"NoBadWords",
"type":"opportunity"},

                    {"id":"", "name":"No PII", "type":"opportunity"},

                    {"id":"", "name":"No PCI", "type":"threat"},

                    {"id":"", "name":"No PHI", "type":"opportunity"},

                    {"id":"", "name":"Completeness",
"type":"opportunity"},

                    {"id":"", "name":"NoToxicity", "type":"threat"},

                    {"id":"", "name":"Accuracy", "type":"threat"},

                    {"id":"", "name":"Robustness",
"type":"opportunity"},
```

```
                    {"id":"", "name":"FactCheck",
"type":"opportunity"},

                    {"id":"", "name":"NoStereotyping", "type":"threat"}

                ]

            },

            {"id":"3", "name":"Transcription",

                "measures":[

                    {"id":"", "name":"TextQuality",
"type":"opportunity"},

                    {"id":"", "name":"CoSelection",
"type":"opportunity"},

                    {"id":"", "name":"UtilityScore", "type":"threat"},

                    {"id":"", "name":"NoBadWords",
"type":"opportunity"},

                    {"id":"", "name":"No PII", "type":"opportunity"},

                    {"id":"", "name":"No PCI", "type":"threat"},

                    {"id":"", "name":"No PHI", "type":"opportunity"},

                    {"id":"", "name":"WecLevel", "type":"opportunity"},

                    {"id":"", "name":"WordLossLevel", "type":"threat"},

                    {"id":"", "name":"Completeness",
"type":"opportunity"},

                    {"id":"", "name":"NoToxicity", "type":"threat"},

                    {"id":"", "name":"Accuracy", "type":"threat"},

                    {"id":"", "name":"Robustness",
"type":"opportunity"},

                    {"id":"", "name":"FactCheck",
"type":"opportunity"},

                    {"id":"", "name":"NoStereotyping", "type":"threat"}

                ]

            },

            {"id":"5", "name":"Q&A",
```

```
                "measures":[

                    {"id":"", "name":"TextQuality",
"type":"opportunity"},

                    {"id":"", "name":"CoSelection",
"type":"opportunity"},

                    {"id":"", "name":"UtilityScore", "type":"threat"},

                    {"id":"", "name":"NoBadWords",
"type":"opportunity"},

                    {"id":"", "name":"No PII", "type":"opportunity"},

                    {"id":"", "name":"No PCI", "type":"threat"},

                    {"id":"", "name":"No PHI", "type":"opportunity"},

                    {"id":"", "name":"Completeness",
"type":"opportunity"},

                    {"id":"", "name":"NoToxicity", "type":"threat"},

                    {"id":"", "name":"Accuracy", "type":"threat"},

                    {"id":"", "name":"Robustness",
"type":"opportunity"},

                    {"id":"", "name":"FactCheck",
"type":"opportunity"},

                    {"id":"", "name":"NoStereotyping", "type":"threat"}

                ]

            },

                       {"id":"7", "name":"Text Generation",

                "measures":[

                    {"id":"", "name":"TextQuality",
"type":"opportunity"},

                    {"id":"", "name":"CoSelection",
"type":"opportunity"},

                    {"id":"", "name":"UtilityScore", "type":"threat"},

                    {"id":"", "name":"NoBadWords",
"type":"opportunity"},

                    {"id":"", "name":"No PII", "type":"opportunity"},
```

```
                    {"id":"", "name":"No PCI", "type":"threat"},

                    {"id":"", "name":"No PHI", "type":"opportunity"},

                    {"id":"", "name":"Completeness",
"type":"opportunity"},

                    {"id":"", "name":"NoToxicity", "type":"threat"},

                    {"id":"", "name":"Accuracy", "type":"threat"},

                    {"id":"", "name":"Robustness",
"type":"opportunity"},

                    {"id":"", "name":"FactCheck",
"type":"opportunity"},

                    {"id":"", "name":"NoStereotyping", "type":"threat"}

                ]

            }

        ]
```

3. Proposed Enhancements

To enable a more flexible, maintainable system, we propose the following capabilities:

3.1 Add New Models with Selected Features

Requirement:
Allow the addition of new models (e.g., "Anthropic", "Mistral") and the ability to assign any subset of features to each model.

Behavior:
When a feature is assigned to a model, all the default scores associated with that feature should automatically link to the model-feature pair.

Adding New Model can be a complicated task, Need more discussion & clarification.

3.2 Add New Features to Existing Models

Requirement:
Allow addition of new features (e.g., "Summarization V2", "Entity Detection") and associate them with one or more existing models.

Behavior:
During assignment, we can select which scores are applicable to that feature per model.

3.3 Custom Score Assignment per Feature

Requirement:
Allow fine-grained control to modify which scores are assigned to which features — independently of the default or existing templates.

Use Case:
If "Summarization" usually has 13 scores, a model might only use a subset, or may include custom-defined scores.

##New Score can be added only through STUDIO portal.

3.4 Score-Feature Flexibility

Requirement:
- A score (e.g., "FactCheck") can be assigned to any feature.
- Scores are not restricted to predefined templates.
- Scores can be removed or replaced for a given feature.

Example:
Assign Accuracy to Summarization for Model A but not for Model B.

4. Data Structure Recommendation (JSON)

To support these dynamic relationships, we suggest the following hierarchical structure:

```
{
  "models": [
    {
      "name": "Open AI",
      "features": [
        {
          "name": "Summarization",
          "scores": [
            {"name": "TextQuality", "type": "opportunity"},
            {"name": "Accuracy", "type": "threat"}
          ]
        },
        {
          "name": "Translation",
          "scores": [ ... ]
        }
      ]
    },
    {
      "name": "Google AI",
      "features": [
        ...
      ]
    }
```

```
    ]
}
```

Requirement Document: Reflective Testing - New Evaluation Modes

1. Introduction

Enhance the existing Reflective Testing interview creation screen to support two new Evaluation Modes — **User Defined Questions** and **Default Questions**, in addition to the existing **Core Model** and **Project Specific** modes. The UI should adjust dynamically based on the selected Evaluation Mode.

2. Current State

The current Reflective Testing interface allows users to configure an interview using the following fields:

Topic: Dropdown (Existing options: Bias, Ethics, Explainability, Fact Checking, Hallucination, Inconsistency, Safety)

Evaluation Mode: Dropdown (Existing options: Core Model, Project Specific)

Domain Tag(s): Dropdown

Interview Model: Search Box

Question Type: Dropdown

No. of Questions: Number input with increment/decrement buttons

The workflow depends on the selections made in these fields.

3. Proposed Changes

3.1. Update Evaluation Mode Options

Modify the "Evaluation Mode" dropdown list.

The dropdown should now contain the following four options:

Core Model (Existing)

Project Specific (Existing)

Default Questions (New)

User Defined Questions (New)

3.2. Workflow for "Default" Evaluation Mode

Trigger: User selects any Topic AND selects Evaluation Mode = "Default"

UI Behavior:

The following fields become hidden or disabled:

Domain Tag(s)

Interview Model

The following fields remain visible and enabled:

Topic (already selected)

Evaluation Mode (already selected)

Question Type

No. of Questions

User Action: The user can select the desired Question Type(s) and specify the No. of Questions.

Final Step: The user clicks the "Create Interview" button to proceed.

3.3. Workflow for "User Defined Questions" Evaluation Mode

Trigger: User selects any Topic AND selects Evaluation Mode = "User Defined Questions".

UI Behavior:

The following fields become hidden or disabled:

Domain Tag(s)

Interview Model

Question Type

No. of Questions

The following fields/components become visible and enabled:

Topic (already selected)

Evaluation Mode (already selected)

File Upload Component: A standard file upload button/area.

Help Guide Link/Text: Text or a link placed near the file upload component (e.g., "Need help with formatting? See guide.") providing access to instructions on the required file format.

User Action: The user uploads a file containing their custom questions (formatted according to the provided guide).

Final Step: The user clicks the "Create Interview" button to proceed using the questions from the uploaded file.

3.4. Workflow for Existing Evaluation Modes ("Core Model" & "Project Specific")

The user interface and workflow for selecting Evaluation Mode = "Core Model" or Evaluation Mode = "Project Specific" remain unchanged from the current implementation. All fields (Topic, Evaluation Mode, Domain Tag(s), Interview Model, Question Type, No. of Questions) should function as they do currently for these modes.

4. Additional Requirements

File Format Guide: Create a clear help guide or documentation page explaining the required file format (e.g., Text, JSON) and structure for the "User Defined Questions" upload. This guide should be easily accessible from the interface, as described in section 3.3.

Error Handling: Implement basic validation for the file upload (e.g., checking if a file is selected before allowing "Create Interview" to be clicked in the User Defined Questions mode). More robust format validation should occur on the backend upon processing the file.

5. UI Notes

Fields should dynamically show/hide based on the Evaluation Mode selected.

The screen should remain clean and intuitive, updating instantly with mode changes.

Validation should ensure:

Required fields are filled.

File format is correct for user-defined questions.

6. Acceptance Criteria

Test Case	Condition	Expected Result

1	Topic selected, Evaluation Mode = "Default"	Only Question Type & No. of Questions visible
2	Topic selected, Evaluation Mode = "User Defined Questions"	Only File Upload field visible
3	Evaluation Mode = Core Model / Project Specific	All fields visible and functional
4	File uploaded in incorrect format	Show validation error
5	Help guide clicked	Sample format is shown/downloaded