ESG Checklist AI Automation Project – Documentation Plan

Objective, Scope, and Value Proposition

Objective: Enable the Risk & Assurance department at e& to efficiently manage and verify ESG (Environmental, Social, Governance) compliance through an AI-driven, automated checklist system. This project aims to streamline sustainability auditing by providing a structured tool for collecting, analyzing, and reporting ESG data. The solution will ensure consistent data capture, timely risk notification, and data-driven scoring of free-text responses to support regulatory compliance.

Scope: The system will encompass the end-to-end workflow of ESG checklist management: defining checklist items, assigning them to owners, collecting answers, automatically scoring answers via NLP, tracking item status, and generating summary reports. It will integrate with e&'s corporate systems (e.g. identity management, messaging) and include administrative functions (e.g. creating checklist templates, configuring scoring criteria). Out of scope are unrelated Risk & Assurance functions and non-ESG compliance tasks.

Value Proposition: This automation delivers clear business benefits:

- **Efficiency:** Automating answer collection and using AI to score open-text responses drastically reduces manual review effort. Routine tasks (data entry, initial analysis) become faster, freeing teams to focus on high-value assurance activities.
- **Consistency & Accuracy:** By enforcing mandatory fields and applying NLP scoring, the system ensures uniform treatment of all checklist items. Ambiguous or incomplete answers are flagged systematically. This improves audit quality and compliance accuracy.
- **Risk Reduction:** Early detection of missing or insufficient responses and automated owner notifications help mitigate compliance gaps before they escalate. Proactive alerts and dashboards keep stakeholders informed.
- **Data-Driven Insights:** Consolidating responses and scores into dashboards enables trend analysis (e.g. recurring risk areas) and supports strategic decision-making. ESG software is recognized as a strategic management instrument for sustainability goals.
- **Auditability:** All data and actions are logged (audit trail) to support internal/external audits. Structured data collection ensures traceability, as emphasized by ESG reporting best practices.

Users and Stakeholders

- **End Users:**
- **Checklist Owners (Department Employees):** Business unit personnel responsible for providing answers to assigned ESG items.
- **Risk/Assurance Reviewers:** Internal auditors or risk officers who review submitted checklists, approve or request revisions.

- **System Administrator:** IT or ESG coordinator who manages checklist templates, user roles, and scoring configurations.

- **Stakeholders:**

- **Risk & Assurance Management:** Department leadership relying on accurate ESG compliance data to assess enterprise risk.
- **e& Executive Committee:** Senior executives (CRO, CIO, etc.) interested in governance and compliance reporting outcomes.
- **IT Security/Infrastructure Team:** Ensures the system meets corporate security, network, and deployment standards.
- **Regulatory/Compliance Officers:** Need reliable ESG data to satisfy regulatory requirements and external audits.
- **External Auditors/Consultants:** May access reports or summaries for compliance verification.

Assumptions and Constraints

- **Assumptions:**
- Users will have corporate credentials and access to e& single-sign-on (SSO) systems.
- Relevant ESG checklist content (questions, scoring rubrics) will be provided by risk subject-matter experts.
- Sufficient historical or reference data exists to train or calibrate the NLP scoring model.
- Stakeholders will adopt the new system and workflows, with training provided for all user roles.

- Integration endpoints (e.g. email server for notifications, identity provider, database) are available in the e& environment.

- **Constraints:**

- **Security & Compliance:** The application must comply with e& security policies (e.g., data encryption at rest/in transit, access controls).
- **Technology Stack:** Development is limited to approved technologies (e.g., Java/.NET, Angular, on-prem databases, etc.) as per e& IT standards.
- **Performance:** The system should support expected load (e.g., hundreds of concurrent users during peak audit cycles) without degradation.
- **Data Privacy:** No unauthorized personal data (PII) will be stored; sensitive answers must be protected per corporate policy.
- **Regulatory Changes:** The checklist content may change with new regulations; the system must be easily updatable.
- **Budget and Timeline:** Resource and time constraints may limit the use of large-scale AI models; a pragmatic, “Just Barely Good Enough” approach is expected.

Functional Requirements

Functional requirements are specific product features that enable users to achieve their goals. The system shall provide the following capabilities:

- **Authentication & Authorization:** The system shall require all users to log in using e& corporate SSO credentials, enforcing role-based access control.
- **Dashboard:** The system shall present each user with a personalized dashboard listing their active ESG checklists (with status indicators) immediately upon login.
- **Checklist Navigation:** The system shall allow users to navigate through all checklist items, organized by category or topic, with clear indications of which fields are mandatory.
- **Answer Entry:** The system shall allow checklist owners to enter or update answers for each item (free text, numeric data, or file attachments). Mandatory fields must be validated before allowing submission.
- **Auto-Save Drafts:** The system shall periodically auto-save users' in-progress answers to prevent data loss, and allow users to manually save progress.
- **Submit for Review:** Once all mandatory items are filled, the user shall be able to submit the checklist. Upon submission, the system shall lock the checklist (read-only) and change its status to "Under Review."
- **Automated Scoring:** Immediately upon submission, the system shall invoke the AI/NLP engine to analyze free-text answers and assign a compliance score (0–1) to each item, based on predefined criteria.
- **Feedback Display:** After scoring, the system shall display each item's score and (optionally) AI-generated feedback or suggestions to the user and reviewer.
- **Reviewer Actions:** The system shall allow authorized reviewers to view submitted checklists, add comments to specific items, and either approve the checklist or request revisions (reject).
- **Status Updates:** Upon approval or rejection, the system shall update the checklist status (e.g. "Approved" or "Returned for Revision") and trigger notifications to the appropriate users.
- **Notifications:** The system shall send email/in-app notifications to checklist owners when checklists are submitted, approved, or rejected, and to remind owners of upcoming deadlines (configurable).
- **Reminders:** The system shall automatically notify owners of any overdue checklists or items, according to predefined schedules.
- **Reporting:** The system shall provide a reporting module where managers can generate ESG compliance summaries (e.g. average scores by department, completed checklist counts) and export reports in PDF or CSV format.
- **Search and Filter:** The system shall allow users to search for checklists by keyword, filter lists by status or owner, and sort results by date or priority.
- **Data Import/Export:** The system shall allow administrators to import a new checklist template from a file (e.g. JSON or Excel) and export existing checklist data for archival or analysis.
- **Audit Log:** The system shall record all significant user actions (login, answers submitted, approvals, etc.) in an immutable audit log, viewable by administrators.
- **Configurable Workflows:** The system shall allow configuration of workflow rules, such as setting different reviewer roles for different checklist types or items.
- **User Management:** The system shall allow administrators to assign users to roles (Owner, Reviewer, Admin) and manage access groups for checklist assignments.
- **Role-Based Interfaces:** The system shall present user interfaces tailored to each role (e.g., owners see answer fields, reviewers see read-only answers plus action buttons).

- **Integration:** The system shall integrate with corporate IT systems (e.g. Active Directory for user roles, SMTP for email) to leverage existing infrastructure.
- **Help/Guidance:** The system shall provide context-sensitive help or tooltips for each checklist item and a glossary of key terms for user reference.

Each requirement is driven by a use case or user need (for example, “submit checklist” or “receive notification”) and is stated in clear, testable terms. The phrasing follows industry conventions (e.g. “The system shall…”) to ensure clarity and accountability.

Non-Functional Requirements

Non-functional requirements impose constraints on system operation and quality. The key NFRs for this project include:

- **Performance:** The system shall respond to user interactions within 2 seconds for normal operations (e.g. loading a checklist page) and complete NLP scoring within 10 seconds per item.
- **Scalability:** The design shall support scaling up to the full size of e& (multiple business units), supporting up to 1,000 concurrent users without degradation.
- **Reliability/Availability:** The system shall achieve 99.9% uptime during business hours. Critical services (login, data save, scoring) must have high availability.
- **Security:** All data in transit must use TLS encryption. Sensitive data (e.g. answer text) shall be encrypted at rest. The system shall implement OWASP security best practices to prevent common web vulnerabilities.
- **Compliance:** The system shall comply with relevant standards (e.g. ISO 27001) and regional regulations regarding data protection (if applicable), as determined by e&’s policies.
- **Logging and Monitoring:** The system shall produce detailed logs of errors and usage metrics. Automated monitoring and alerting shall be in place for key failures (e.g. scoring engine downtime).
- **Usability:** The user interface shall be clean and intuitive, with support for English (and if needed regional language) and accessibility guidelines (e.g. high-contrast modes). Minimal training should be required for end users.
- **Maintainability:** Code and configuration should be modular. Documentation (code comments, design docs) shall be updated. The system shall allow future extension (e.g. adding new question types or AI models) with minimal impact.
- **Backup and Recovery:** The system shall regularly back up its database. Recovery procedures shall allow the system to be restored to a point no more than 24 hours prior in case of failure.
- **Interoperability:** The system shall use standard data formats (JSON, CSV) and APIs to allow future integration with other tools (e.g. BI/reporting platforms).
- **Capacity Planning:** Storage and processing capacity shall be sufficient to store at least 3 years of checklist records per item and handle spikes in data entry around audit deadlines.
- **Localization:** The system shall support regional date/time formats and number formats as per user locale settings.
- **Browser Support:** The web interface shall be compatible with modern browsers (Chrome, Edge, Firefox) as approved by e&’s IT department.
- **Documentation:** User manuals and technical documentation shall be provided (both inline help and separate guides) as part of the deliverables.
- **Response Time:** The system shall maintain end-to-end response times under defined SLAs; slowdowns beyond thresholds should trigger alerts.

These NFRs ensure the system's quality and compliance. For example, by specifying "secure login" and "data encryption" the system satisfies corporate security policies, while "99.9% uptime" addresses availability expectations.

Use Case Descriptions

The following use cases capture the key interactions between users and the system. Each is described with **ID, Name, Description, Actors, Preconditions, Postconditions, Main Flow,** and **Alternate Flows** as recommended. Main flows are numbered sequential steps; alternate flows refer back to those steps.

- **Use Case UC1: User Login**
- **ID:** UC1
- **Name:** User Login
- **Description:** An authorized user logs into the system to gain access to their dashboard.
- **Actors:** Checklist Owner or Reviewer (Employee)
- **Preconditions:** User has valid corporate credentials and access rights.
- **Postconditions:** User is authenticated, a session is established, and the main dashboard is displayed.
- **Main Flow:**
    1. User navigates to the application URL.
    2. System prompts for SSO or corporate credentials.
    3. User enters credentials and submits.
    4. System verifies credentials via corporate identity provider.
    5. System opens the user's dashboard (showing assigned checklists).
- **Alternative Flows:**
    - **Alt 1:** If authentication fails at step 4 (wrong credentials or account locked), the system displays an error message and returns to the login screen (refer to step 2).
- **Use Case UC2: View Checklist Dashboard**
- **ID:** UC2
- **Name:** View Checklist Dashboard
- **Description:** A logged-in user views an overview of their assigned checklists and statuses.
- **Actors:** Checklist Owner, Reviewer
- **Preconditions:** User is logged in (from UC1) and has one or more checklist assignments.
- **Postconditions:** Checklist list is displayed; user can select an item to view or edit.
- **Main Flow:**
    1. System retrieves list of checklists assigned to the user.
    2. System displays a table or cards for each checklist (showing title, status, due date).
    3. User selects a checklist to proceed (e.g., clicking its title).
- **Alternative Flows:**
    - **Alt 1:** If the user has no assignments, the system shows a "No checklists" message.
- **Use Case UC3: Start/Initialize Checklist**

• **ID:** UC3
• **Name:** Initialize Checklist
• **Description:** A checklist owner opens a specific checklist to begin entering responses.
• **Actors:** Checklist Owner
• **Preconditions:** Checklist exists and status is *Draft* or *Returned for Revision*; user has access rights.
• **Postconditions:** Checklist item detail view is displayed; user can enter answers.
• **Main Flow:**
    1. User clicks on a checklist from the dashboard (UC2).
    2. System loads the checklist items and current answers (if any).
    3. System displays each question with input fields for answers.

• **Alternative Flows:**

    ◦ **Alt 1:** If the checklist status is *Approved*, the system opens it in read-only mode (user cannot edit).
    ◦ **Alt 2:** If the checklist data fails to load (error), the system shows an error message and offers to retry.

• **Use Case UC4: Answer/Edit Checklist Item**

• **ID:** UC4
• **Name:** Answer Checklist Item
• **Description:** The checklist owner enters or modifies the answer for an individual checklist question.
• **Actors:** Checklist Owner
• **Preconditions:** The checklist is open in edit mode (UC3).
• **Postconditions:** The answer (and any attachments) is saved for that item; mandatory checks are applied.
• **Main Flow:**
    1. User locates a checklist question and clicks into its answer field.
    2. User types or selects an answer (text entry, number, or file upload).
    3. User clicks “Save” or moves to the next question.
    4. System validates the answer (e.g. mandatory field not empty).
    5. System saves the answer to the database and marks the item as “Answered.”

• **Alternative Flows:**

    ◦ **Alt 1:** If the answer is left blank for a mandatory field (detected at step 4), the system highlights the field in red and displays a warning, preventing saving until resolved (returns to step 2).
    ◦ **Alt 2:** If the system cannot save (e.g. network error), it notifies the user and retries saving.

• **Use Case UC5: Auto-Save Progress**

• **ID:** UC5
• **Name:** Auto-Save Progress
• **Description:** The system automatically saves in-progress answers periodically to prevent data loss.
• **Actors:** System (Background Service)
• **Preconditions:** A checklist is open and being edited (UC3/UC4).

- **Postconditions:** Any new answers entered since the last save are committed to the database.
- **Main Flow:**
    1. Timer triggers auto-save at predefined intervals (e.g. every 2 minutes).
    2. System captures the current state of all answer fields.
    3. System sends the data to the server.
    4. Server saves the answers and returns a confirmation.

- **Alternative Flows:**

    - **Alt 1:** If the server is unreachable at step 3, the client queues the data and retries on next interval.

- **Use Case UC6: Submit Checklist for Review**

- **ID:** UC6
- **Name:** Submit Checklist
- **Description:** The checklist owner finalizes all answers and submits the checklist to the reviewer.
- **Actors:** Checklist Owner
- **Preconditions:** All mandatory items have non-empty answers; checklist status is *Draft* or *In Progress*.
- **Postconditions:** Checklist status changes to *Submitted/Under Review*; system triggers scoring and notifications.
- **Main Flow:**
    1. User clicks the “Submit” button.
    2. System validates that all mandatory items are answered.
    3. System locks the checklist (no further edits).
    4. System invokes the AI scoring engine for all free-text answers.
    5. System records the scores and updates each item’s status.
    6. System changes the checklist status to *Under Review*.
    7. System sends notification to the assigned reviewer(s) (see UC7).

- **Alternative Flows:**

    - **Alt 1:** If any mandatory answer is missing at step 2, the system aborts submission, highlights the missing field(s), and prompts the user to complete them (return to step 1).
    - **Alt 2:** If the AI engine fails at step 4, the system still submits the checklist but logs an error and notifies administrators.

- **Use Case UC7: AI Analyze Answers (Automated Scoring)**

- **ID:** UC7
- **Name:** AI/NLP Answer Scoring
- **Description:** Upon submission, the system analyzes each text answer using NLP to produce a compliance score.
- **Actors:** System (AI Module)
- **Preconditions:** The checklist has been submitted (UC6) with answers in place.
- **Postconditions:** Each checklist item is annotated with a score (e.g., 0–1) and optional feedback.
- **Main Flow:**
    1. System sends each answer text to the NLP scoring service.

2. NLP service computes semantic similarity and other metrics to generate a score.
3. NLP service returns scores (and any suggested flags) to the system.
4. System updates each item's record with the score and optional comment.

• **Alternative Flows:**

  ◦ **Alt 1:** If an answer is too short or empty, the NLP service returns a score of 0.0 (and flags it as non-compliant).
  ◦ **Alt 2:** If the NLP service is temporarily unavailable, the system queues the checklist and retries scoring later.

• **Use Case UC8: Notify Owner of Missing/Required Info**

• **ID:** UC8
• **Name:** Owner Notification
• **Description:** The system automatically notifies the checklist owner when they need to take action (e.g. submission reminder, revision request).
• **Actors:** System
• **Preconditions:** Trigger conditions are met (e.g., approaching deadline or reviewer action).
• **Postconditions:** An email or in-app alert is sent to the owner.
• **Main Flow:**
  1. System detects an event (e.g., a checklist is submitted, or a deadline is near).
  2. System formats a notification message (including checklist ID and action needed).
  3. System sends the notification to the owner's email and/or in-app notification center.

• **Alternative Flows:**

  ◦ **Alt 1:** If the email fails to send (bounced), the system logs the failure and displays a non-critical warning in the admin panel.

• **Use Case UC9: Reviewer Views Submitted Checklist**

• **ID:** UC9
• **Name:** View Submitted Checklist
• **Description:** A reviewer (Risk Officer) opens a submitted checklist to inspect answers.
• **Actors:** Reviewer (Risk Officer)
• **Preconditions:** Checklist status is *Under Review*; reviewer is assigned and logged in.
• **Postconditions:** Reviewer sees all answers and scores; they can add comments.
• **Main Flow:**
  1. Reviewer selects the submitted checklist from their dashboard.
  2. System loads the checklist in read-only mode.
  3. System displays each question, user's answer, AI score, and any previous comments.
  4. Reviewer adds their review comments and selects "Approve" or "Request Revision."

• **Alternative Flows:**

  ◦ **Alt 1:** If the reviewer is not authorized for this checklist, the system denies access and shows an error.

• **Use Case UC10: Approve Checklist**

• **ID:** UC10
• **Name:** Approve Checklist
• **Description:** The reviewer confirms the checklist is satisfactory and marks it approved.
• **Actors:** Reviewer
• **Preconditions:** The reviewer has viewed the checklist and added any comments (UC9).
• **Postconditions:** Checklist status is set to *Approved*; final notification is sent.
• **Main Flow:**
    1. Reviewer clicks “Approve.”
    2. System updates checklist status to *Approved*.
    3. System records reviewer name and timestamp.
    4. System sends an approval notification to the owner.

• **Alternative Flows:**

    ◦ *None.* (Approval is a terminal action.)

• **Use Case UC11: Return Checklist for Revision**

• **ID:** UC11
• **Name:** Reject/Revise Checklist
• **Description:** The reviewer finds issues and sends the checklist back to the owner for correction.
• **Actors:** Reviewer
• **Preconditions:** The reviewer is reviewing the checklist (UC9).
• **Postconditions:** Checklist status changes to *Returned for Revision*; owner is notified.
• **Main Flow:**
    1. Reviewer clicks “Request Revision,” enters revision comments, and submits.
    2. System updates status to *Returned for Revision*.
    3. System saves reviewer comments on relevant items.
    4. System sends a notification to the owner indicating required changes.

• **Alternative Flows:**

    ◦ *None.* (Upon return, the flow essentially goes back to UC3 for owner rework.)

• **Use Case UC12: Manage Checklist Items (Admin)**

• **ID:** UC12
• **Name:** Manage Checklist Items
• **Description:** An administrator creates, edits, or deactivates questions in the ESG checklist template.
• **Actors:** System Administrator
• **Preconditions:** Admin is logged in with appropriate privileges.
• **Postconditions:** The checklist template is updated; changes apply to future checklists.
• **Main Flow:**
    1. Admin navigates to the “Checklist Designer” interface.
    2. System displays the list of current checklist questions (with IDs, text, mandatory flag).

3. Admin can add a new question, modify existing text, change mandatory status, or delete a question.
4. Admin saves the changes.
5. System validates the template (e.g. no duplicate IDs, required fields are set).
6. System updates the checklist schema in the database.

- **Alternative Flows:**
    - **Alt 1:** If the admin tries to delete a question that is already used in a past checklist, the system either disallows deletion or marks it inactive (per policy).

- **Use Case UC13: Configure Scoring Criteria (Admin)**

- **ID:** UC13
- **Name:** Configure Scoring
- **Description:** An administrator defines or adjusts how AI scores answers (e.g. thresholds, keywords).
- **Actors:** System Administrator
- **Preconditions:** Admin has access to scoring configuration settings.
- **Postconditions:** New scoring rules are saved and used for future analyses.
- **Main Flow:**
    1. Admin selects “Scoring Settings.”
    2. System displays current scoring thresholds (e.g. keyword weights, similarity cutoffs).
    3. Admin edits weights, adds key terms for scoring, or adjusts point allocations.
    4. Admin saves the configuration.
    5. System validates the new settings.
    6. System applies changes for subsequent NLP analyses.

- **Alternative Flows:**
    - **Alt 1:** If an invalid configuration is entered (e.g. non-numeric weight), the system shows an error and rejects the update.

- **Use Case UC14: Generate Compliance Report**

- **ID:** UC14
- **Name:** Generate Report
- **Description:** A user generates a summary report of ESG compliance across checklists (for a team, department, or time period).
- **Actors:** Manager, Risk Officer
- **Preconditions:** There exist completed checklists in the system.
- **Postconditions:** A report is produced (on-screen or exported) showing aggregated scores and metrics.
- **Main Flow:**
    1. User opens the “Reporting” section.
    2. System prompts for report parameters (date range, department, checklist type).
    3. User enters criteria and clicks “Run Report.”
    4. System aggregates relevant checklist data (e.g. computes average scores per category).
    5. System displays the report on-screen and offers options to export (PDF/CSV).

- **Alternative Flows:**
    - **Alt 1:** If no data matches criteria, the system informs the user and offers to adjust filters.

- **Use Case UC15: Export Checklist Data**

- **ID:** UC15
- **Name:** Export Data
- **Description:** An administrator or auditor exports raw checklist data for offline analysis or archive.
- **Actors:** System Administrator, Auditor
- **Preconditions:** Relevant checklists exist in the system.
- **Postconditions:** A file (e.g. CSV or Excel) containing checklist data is downloaded or emailed.
- **Main Flow:**
    1. User selects “Export” from the menu.
    2. System asks for export scope (all checklists, date range, specific items).
    3. User confirms export.
    4. System generates the data file in the chosen format.
    5. System prompts user to download the file or sends it via email.

- **Alternative Flows:**
    - **Alt 1:** If the export request exceeds system limits, the system splits the data into multiple files or requests narrower filters.

- **Use Case UC16: Manage Users and Roles**

- **ID:** UC16
- **Name:** Manage User Roles
- **Description:** An administrator assigns or updates user roles (e.g. Owner, Reviewer, Admin) to control access and permissions.
- **Actors:** System Administrator
- **Preconditions:** Admin is logged in with privileges.
- **Postconditions:** User accounts are updated with new roles/groups.
- **Main Flow:**
    1. Admin navigates to “User Management.”
    2. System lists all users and their current roles/groups.
    3. Admin selects a user and assigns or revokes roles (e.g. granting reviewer rights).
    4. Admin saves changes.
    5. System updates the user’s permissions accordingly.
- **Alternative Flows:**
    - **Alt 1:** If the admin tries to assign an invalid role, the system flags an error and aborts that change.

(Use Case flows reference main steps by number. For example, in UC1 Alt 1, the system returns to Main Flow step 2.) Each use case can be captured in a diagram (Use Case Diagram) to show actors and these interactions, as described below.

System Design and Diagrams

The system's design will be documented with UML diagrams and guidelines. Use standard UML 2.x notation and tools (e.g. PlantUML, draw.io, Lucidchart) to create the following views. Diagrams should follow best practices (clear naming, minimal crossing lines, consistent layout).

Use Case Diagram

Use case diagrams provide an overview of the system's functionality and its actors. In the use case diagram, draw each **actor** (e.g. *Checklist Owner*, *Reviewer*, *Admin*) as a stick figure on the perimeter, and each **use case** (from the list above) as an ellipse within the system boundary box. Associate actors to the use cases they initiate. Use strong verb phrases for use case names (e.g. "Submit Checklist," "Approve Checklist"). Place primary actors on the left and use cases on the right for a natural reading order. For clarity, avoid including alternative flow details in this diagram – focus only on primary interactions and major extensions (uses or extends relationships if needed). Use stereotypes (e.g. «includes», «extends») only when they add meaningful context.

Notation & Tools: Standard UML notation (solid lines for associations, include and extend arrows where applicable). Draw the system boundary as a labeled rectangle (e.g. "ESG Checklist System"). Tools like PlantUML or draw.io can generate these diagrams textually or via drag-and-drop; ensure font sizes and layout keep the diagram legible.

Class Diagram

Class diagrams model the static structure of the system. Identify the core classes and their responsibilities. Key classes might include: **Checklist** (attributes: id, title, status, owner, etc.; methods: submit(), calculateScore()), **ChecklistItem** (attributes: question, answer, score, isMandatory), **User** (attributes: id, name, role), **Notification**, **ReportGenerator**, **AIEngine**, and **Attachment**. Each class should encapsulate its data (attributes) and behavior (methods). For example, *Checklist* may have methods to add answers, submit for review, and compute an overall score (which internally calls AIEngine). *User* might have methods like sendNotification().

Figure: Example UML class notation (name, attributes, operations). The above illustrates UML class notation. In our diagram, use this notation: class name in the top compartment, attributes (with types) in the middle, and operations (methods) in the bottom. Mark visibility with + or – as needed (most fields can be private, exposing only needed getters/setters). Show relationships: e.g., a Checklist "has many" ChecklistItems (composition), a User *creates* or *reviews* a Checklist. If there are clear hierarchies, use generalization (inheritance). Otherwise, favor association arrows with role names or multiplicities. **Responsibilities:** Each class should follow the single-responsibility principle—e.g., *AIEngine* handles text scoring (it should not manage user sessions), *Notification* handles message formatting, etc.

Explanation of [29]: This image shows a class named *Class* with attributes and methods. It exemplifies the notation for classes, attributes, and operations. When drawing our actual diagram, replace "Class" with domain-specific names (e.g. *Checklist*, *User*). Ensure logical grouping: for instance, link *ChecklistItem* instances to their parent *Checklist*, and show that *Notification* depends on *User* and *Checklist*.

Sequence Diagrams

Sequence diagrams depict object interactions over time for specific scenarios. We will create at least two sequence diagrams for key flows, such as: **(1) Submit Checklist** and **(2) Approve Checklist**. Each diagram is a vertical lifeline for each participant (actor or class object) with arrows showing method calls or messages between them in chronological order.

- **Scenario 1 – Submit Checklist:** Actors/objects: *User*, *ChecklistController (UI)*, *ChecklistService*, *AIEngine*, *NotificationService*. The flow: user -> ChecklistController:submitChecklist(), ChecklistController -> ChecklistService:validateAndLock(), ChecklistService -> AIEngine:scoreAnswers(), AIEngine -> ChecklistService:returnScores(), ChecklistService -> NotificationService:notifyReviewer(), etc.
- **Scenario 2 – Approve Checklist:** Actors/objects: *Reviewer*, *ReviewController*, *ChecklistService*, *NotificationService*. Flow: reviewer -> ReviewController:approve(), which calls ChecklistService:updateStatus(Approved), then ChecklistService->NotificationService:notifyOwner().

The diagrams should be read left-to-right in message order. Use participant names consistent with class diagram names. For clarity, keep the messages as concise method calls (e.g. `calculateScore()`, `sendEmail()`). Follow the guidelines: name participants consistently, justify message names beside arrows, and include any return messages only if important. Place human/actor entities on the left and system components to the right. The example below (from an enrollment scenario) illustrates the style:

Figure: Sample UML sequence diagram (example of user enrolling scenario). In our diagrams, the lifelines and arrows will similarly trace user interactions and system operations. We should include brief prose (or this caption) explaining the flow if necessary, per best practices.

State Diagram

A state machine (statechart) diagram models the lifecycle of a single Checklist entity in response to events. We will illustrate states such as: **Draft** (initial state) → **In Progress** (as the owner edits) → **Submitted/Under Review** → **Approved** or **Rejected** (final states). You may also include an **Archived** or **Completed** state if needed. Each state represents a phase, and transitions are triggered by events (e.g. *submit*, *approve*, *reject*, *resubmit*).

Rules for the state diagram: use short, descriptive state names. Place the initial (filled circle) at the top-left and final state (bullseye) at the bottom-right for readability. For example: *Draft* → (*submit*) → *Under Review* → (*approve*) → *Approved* (final). If rejected: *Under Review* → (*requestRevision*) → *In Progress*. Do not leave any "black hole" states (states with only incoming or only outgoing transitions). Label transitions with event names in past tense (e.g. "Submitted", "Approved") for clarity. Entry/exit actions can be shown if needed (e.g., on entering *Approved*, trigger a log entry), but can be omitted if they are obvious.

Deployment Diagram

A deployment diagram shows the physical layout of the system components on hardware nodes. For e&'s ESG system, identify the key nodes and artifacts:

- **Nodes (physical):** e.g. *Web Server Node*, *Application Server Node*, *AI/NLP Server Node*, *Database Server Node*, *User Workstation*, etc. (Nodes are typically drawn as 3D boxes.)

- **Artifacts (software):** e.g. “ESGApp.war” on the Application Server, “ChecklistDB” on the Database Node, “NLPService” on the AI Server.
- **Relationships:** Show which artifacts deploy to which nodes (e.g. a “ChecklistService” component artifact on the App Server). Indicate network connections (dashed lines for network) or firewall boundaries as annotations if helpful.

Best practices: label each node with its role (e.g. *Server (Linux)*), and each artifact with its component name. Keep the diagram at a high level: do not detail OS versions or VM configurations. As noted by Miro’s UML guide, use a clear layout and involve stakeholders (e.g. IT/network team) when defining this diagram. Update the diagram when the actual infrastructure is implemented. For example, an arrow from **Web Server Node** to **Application Server Node** would illustrate HTTP communication. Finally, ensure the diagram remains legible and focuses on key elements (omit minor utilities or overly detailed network segments).

Glossary of Terms

- **ESG (Environmental, Social, Governance):** Criteria used to evaluate a company’s sustainability and ethical impact.
- **Checklist Item:** A single compliance question or requirement in the ESG checklist. Each has a question text, an answer field, and may be marked mandatory.
- **Mandatory Field:** An answer field that must be completed before the checklist can be submitted.
- **Owner (of a checklist):** The individual responsible for providing answers to the checklist.
- **Reviewer:** A person (e.g. internal auditor) who examines and approves or rejects completed checklists.
- **NLP Scoring:** The process of using Natural Language Processing algorithms to analyze and score free-text answers based on relevance and completeness.
- **AI Engine:** The subsystem (often a microservice) that executes NLP algorithms and returns an answer score (numeric) and feedback for each question.
- **Compliance Score:** A numeric value (e.g. 0.0–1.0) assigned by the AI engine indicating how well an answer meets expected criteria.
- **Notification:** An automated email or in-app message sent to users (owners or reviewers) to inform them of actions needed (e.g. “Please submit checklist”).
- **Audit Trail:** A log of all significant system events and data changes (e.g. submissions, approvals), used for auditing and traceability.
- **Actor (UML):** An external user or system role that interacts with the system (e.g. *Checklist Owner*, *Risk Officer*).
- **Artifact (Deployment):** A deployable unit of software (e.g. executable, script) that is deployed on a hardware node in a deployment diagram.

Appendix: Example Data Formats and Scoring

Mock Checklist Data (JSON): An example of how checklist data might be structured internally:

```
{
  "checklist_id": "ESG_Q1_2025",
  "title": "Q1 2025 ESG Compliance Checklist",
```

```
  "owner": "jane.doe@eand.com",
  "status": "Draft",
  "items": [
    {
      "id": "ENV001",
      "question": "Have carbon emissions targets been set for 2025?",
      "answer": "Yes, target of a 15% reduction is set for end of 2025.",
      "score": 0.88,
      "comments": "Strong answer with specific target."
    },
    {
      "id": "SOC002",
      "question": "Is there a documented diversity policy?",
      "answer": "",
      "score": 0.00,
      "comments": "No answer provided; mandatory field missing."
    }
  ]
}
```

Expected Output Format (Example): After processing, the system might produce summary data like:

```
{
  "checklist_id": "ESG_Q1_2025",
  "owner": "jane.doe@eand.com",
  "status": "Approved",
  "overall_score": 0.65,
  "item_scores": [
    {"id": "ENV001", "score": 0.88},
    {"id": "SOC002", "score": 0.00}
  ],
  "report": "Some items require follow-up as noted in comments."
}
```

Scoring Criteria (Example): The AI engine's score (0–1) can be interpreted as follows:

- **Score ≥ 0.75:** *Fully Compliant* – Answer meets all essential requirements.
- **Score 0.50 – 0.74:** *Partially Compliant* – Answer covers some requirements but lacks details.
- **Score < 0.50:** *Non-Compliant* – Answer is incomplete or missing key information.

In practice, the scoring algorithm may combine semantic similarity, keyword matching, and other NLP metrics. The system normalizes the result to a 0.0–1.0 scale. For instance, an answer that exactly matches policy language would get a high score, whereas an empty or irrelevant answer yields near 0.00. Thresholds (as above) are configurable by administrators to classify the result.

Note: The above data structures and scoring rubric are illustrative. The final implementation may adjust formats (e.g. CSV instead of JSON for exports) and refine scoring logic based on pilot testing and stakeholder feedback.