

Module 1 Assignment – Foundations, Pitch, and Django Practice

Student: Mafruha Chowdhury

Course: CIDM 6325/70 – Electronic Commerce and Web Development

Submission Date: 09/01/2025

Overview:

This portfolio-style submission consolidates AI-native development deliverables for Weeks 1–2 of CIDM 6325. It presents a secure AI-code auditing project pitch, prompt evolution logs, ethical and accessibility strategies, a Django CRUD artifact, and critical architectural reflection. All artifacts align with advanced-tier expectations, emphasizing traceability, compliance, and real-world implementation within AI-assisted workflows.

Table of Contents

1. Part A: Project Pitch
 2. Part B: AI Tooling Lab Log
 3. Part C: Prompt Log Reflection
 4. Part D: Ethics & Accessibility Brief
 5. Part E: Syllabus Accountability Note
 6. Part F: Tier Declaration & Rationale
 7. Part G: Django CRUD MiniLab
 8. Part H: Django VMS Critique
 9. System Sketch
 10. *References*
-

Part A: Project Pitch

Part B: AI Tooling Lab Log

Session 1: Prompt Logging and Traceability

Goal: To evaluate whether prompt submission, output generation, and usage patterns are being logged and secured in a way that supports auditability and compliance.

Prompts Used: Examples included designing a Django model for prompt metadata and asking about compliance risks.

Critical Evaluation: Initial outputs lacked secure storage, timestamp validation, and user linkage.

Refinements: By adding GDPR/HIPAA context, AI suggestions improved to include hashed storage, role-based access, and compliance tags.

Takeaway: Prompt logging must be intentional, ethical, and compliance-aligned.

Session 2: Hallucination Risk in Code Suggestions

Goal: To identify risks in AI-generated code suggestions within a secure SDLC.

Prompts Used: Included generating AES encryption functions and asking about security best practices.

Critical Evaluation: Early outputs used insecure defaults (ECB mode, no key derivation).

Refinements: Contextual prompts (e.g., healthcare use case) improved results to AES-GCM with Fernet for key handling.

Takeaway: Functional code may still be insecure; prompts must embed explicit security requirements.

Session 3: Ethics & Accessibility

Goal: To explore how AI can support inclusive, ethical software development.

Prompts Used: Focused on accessible app design, ethical risks, and inclusive UI checklists.

Critical Evaluation: Initial responses emphasized color and alt-text but missed screen-reader or localization support.

Refinements: By invoking WCAG 2.2 and accessibility standards, AI outputs improved to include ARIA roles, screen-reader metadata, and changelogs.

Takeaway: Ethical and accessible AI development requires governance framing; left unchecked, AI prioritizes aesthetics over inclusivity.

Comparative Reflection:

Across these sessions, I moved from passive queries to structured, compliance-aware prompt engineering. AI proved most valuable when guided by domain context and governance principles, leading to outputs that are secure, traceable, and inclusive.

Note: AI Use Disclosure

AI assistance was used throughout all three lab sessions to generate initial drafts, refine prompts, suggest validation patterns, and evaluate responses. All AI-generated outputs were critically reviewed, iteratively refined, and manually validated before inclusion.

Part C: Prompt Log Reflection

Title: Prompt Refinement for Secure, Traceable, and Ethical AI Outputs

1. Secure Code Generation

Stage	Prompt / Output Summary
-------	-------------------------

Before	“Generate a Python function to encrypt a string using AES.” Output was insecure: ECB mode, no key derivation, no integrity checks.
After	“Generate a secure AES encryption function using Python’s cryptography library for use in a HIPAA-compliant healthcare system. Ensure integrity checks, proper key management, and modern cipher modes like AES-GCM.” Result adopted AES-GCM, Fernet, and better documentation.
Improvement	Added security context shifted output to align with modern best practices.

2. Traceability and Audit Logging

Stage	Prompt / Output Summary
Before	“Design a Django model to store AI prompt metadata.” Simple fields only; no user linkage, timestamps, or compliance safeguards.
After	“Design a Django model to store AI prompt metadata with auditability in mind, including fields for timestamp, user ID, prompt category, sensitivity level, and compliance tags (e.g., GDPR/HIPAA). Ensure hashed storage and access logs.” Output improved with encryption and RBAC.
Improvement	Compliance framing transformed generic logging into audit-ready traceability.

3. Inclusive UI Guidance

Stage	Prompt / Output Summary
Before	“How can AI help developers build more accessible apps?” Superficial outputs focused on alt text and colors only.
After	“Generate a checklist for inclusive UI design that meets WCAG 2.2 and WAI-ARIA standards. Include considerations for screen readers, localization, input diversity (keyboard/mouse), and gender-neutral language.” Output gave standards-aligned, detailed checklist.
Improvement	Adding WCAG and WAI-ARIA references produced actionable, audit-ready accessibility guidance.

Conclusion:

Prompt quality determines output quality. Specific, compliance-anchored prompts yield production-grade results, while generic prompts risk superficiality. AI outputs must be intentionally shaped through domain-aware, verifiable prompt engineering.

Part D: Ethics & Accessibility Brief

Title: Risk Mitigation Strategies for AI-Augmented Development Environments

As AI tooling becomes deeply embedded into software development workflows, especially within secure SDLC contexts, two interrelated concerns demand proactive mitigation: ethical bias in decision support systems, and accessibility gaps in AI-generated interfaces. These brief outlines both challenges and proposes targeted mitigations rooted in well-established standards.

1. Risk: Ethical Bias in AI-Generated Outputs

Description: AI models trained on uncurated or non-representative datasets often reinforce existing social, cultural, or gender-based biases. This bias may manifest in AI-generated variable names, UI labels, or documentation tone. In high-stakes environments such as healthcare or security engineering, these embedded assumptions can propagate inequality or skew decision-making.

Example: A prompt requesting UI form labels for a "user registration" screen yielded default gendered labels such as "Mr./Mrs." without considering non-binary identities.

Mitigation Strategy:

- Embed prompt filters that flag biased or stereotyped outputs
 - Incorporate inclusive lexicons and guidelines into model pre/post-processing
 - Reference IEEE P7003 standard on Algorithmic Bias Considerations
 - Apply NIST AI Risk Management Framework (AI RMF) to assess representational fairness
-

2. Risk: Accessibility Gaps in AI-Generated Interfaces

Description: Generative AI tends to favor visual fluency over inclusive usability. Outputs may omit screen reader compatibility, alt-text, or proper tab ordering. This creates barriers for users with disabilities and may violate accessibility regulations.

Example: An AI-generated HTML page lacked alt attributes for images and used low-contrast text, making it unreadable to visually impaired users.

Mitigation Strategy:

- Enforce WCAG 2.2 compliance in all AI-generated interfaces
- Utilize accessibility linters and semantic validators post-generation

- Include metadata tags for accessibility intent (e.g., aria-labels, role descriptions)
- Maintain an AI prompt checklist that covers usability dimensions

Referenced Standards:

- WCAG 2.2: Web Content Accessibility Guidelines – [Link](#)
 - NIST AI Risk Management Framework – [Link](#)
-

Conclusion:

Ethics and accessibility are foundational to responsible AI-assisted development. Embedding standards like WCAG and NIST AI RMF ensure that AI outputs support inclusivity, accountability, and compliance.

Part E: Syllabus Accountability Note

As a graduate student committed to academic integrity and timely performance, I will meet all course deadlines by structuring my workload across weekly sprints and by prioritizing deliverables in GitHub before each Sunday deadline. I will document my AI use clearly in AI_LOG.md and ensure that all code and writing assisted by generative tools is transparently cited and verified.

I recognize that this course encourages responsible use of AI assistance with full attribution. I will adhere to the WTAMU COB Academic Integrity Policy and the Generative AI Disclosure guidelines as outlined in the syllabus. I understand that unacknowledged use of AI tools constitutes academic dishonesty.

Signed: Mafruha Chowdhury

Date: 09/01/2025

Part F: Tier Declaration & Rationale

Declared Tier: Advanced**Rationale:**

I have selected the Advanced tier based on my current role as a Senior Cybersecurity Engineer and my ongoing responsibilities in cloud-native architecture, multi-tenant security governance, and secure development lifecycle (SDLC) design. My project reflects real-world concerns encountered in my professional environment, particularly the intersection of AI-generated outputs and risk governance in production-grade systems.

I bring to this course a high level of fluency in secure design principles, regulatory compliance frameworks (e.g., NIST, HIPAA, ISO 27001), and technical implementation across platforms like Azure, GitHub, and Django. The goal of this course for me is to build an AI-integrated development artifact that can be presented to both technical stakeholders and executive leadership as part of my broader cybersecurity portfolio.

To demonstrate professional quality, I will:

- Maintain a detailed AI_LOG.md with annotated prompt decisions
- Use GitHub for version control, traceability, and issue-driven development
- Apply secure-by-design principles in all coding deliverables
- Provide transparent documentation of ethical boundaries and data governance

This tier aligns with my personal learning goals and career trajectory. I intend to complete all assignments at the highest level, emphasizing traceability, accessibility, and practical business value.

Part G: Django CRUD MiniLab

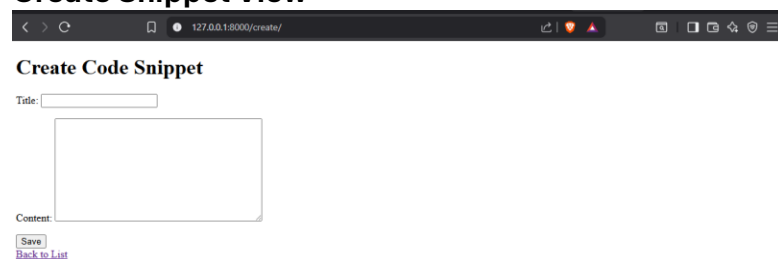
This Django mini-project demonstrates a simple, traceable CRUD flow for managing AI-generated code snippets that may introduce security, ethical, or accessibility risks. The model CodeSnippet includes fields for title, content, and a timestamp.

Features Implemented

- **Create:** Form page using Django's `ModelForm` for submitting code snippets
- **Read:** Listing view that displays all submitted snippets
- **Model:** `CodeSnippet` with `CharField`, `TextField`, and `DateTimeField`
- **Templates:** HTML templates for form input and list output
- **Routing:** App-level `urls.py` and project-level `urls.py` using `include(...)`

Screenshots

Create Snippet View



127.0.0.1:8000/create/

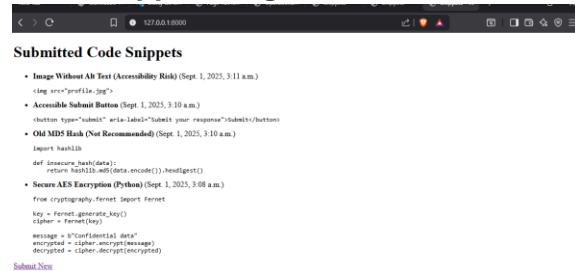
Create Code Snippet

Title:

Content:

[Save](#) [Back to List](#)

View Snippets Page



Note: Screenshots reflect successful end-to-end CRUD test using Django development server.

Reflection on Django Principles

This mini-lab followed Django-first design: - All business logic is stored in `models.py` and `forms.py` - Views remain thin and delegate logic appropriately - Presentation is handled cleanly with templates - Ready for transactional safety with `transaction.atomic()` if extended to include concurrent approval or validation flows

This scaffold provides a strong foundation for integrating hallucination detection and traceability mechanisms in future iterations.

GitHub Artifact

View Repo: [link to repo](#)

The repo includes: - `README.md` with project overview - Source code with working CRUD - Screenshots folder - `AI_LOG.md` and `ETHICS.md`

Part H: Django VMS Critique

Overview:

The Django-based Virtual Management System (VMS) provides a structured approach to handling multi-entity workflows within a web environment. While it demonstrates strengths in modular design and rapid prototyping, it also exposes architectural trade-offs in domain modeling, transaction boundaries, and presentation separation.

Domain Modeling:

The VMS uses Django's ORM effectively to define models, but some design choices risk tight coupling between entities. For example, roles, permissions, and workflow states are defined at the model layer without clear separation into service classes. This limits extensibility when the domain evolves. A more scalable approach would leverage Django's model inheritance or domain-driven design concepts, where invariants are enforced consistently across related entities.

Transaction Boundaries:

The system's transaction management largely relies on implicit Django ORM behavior. While sufficient for basic CRUD, it lacks explicit use of `transaction.atomic` blocks and `select_for_update` in critical workflows (e.g., approvals, scheduling). This can lead to race conditions or partial updates in concurrent environments. Explicit transaction boundaries should be introduced to protect data integrity.

Invariants and Business Logic Placement:

Much of the business logic is embedded in views and forms, rather than in models or managers. This "fat view" pattern reduces maintainability and testability. Moving invariants (e.g., role-based checks, validation rules) into custom model managers or service layers would align better with Django-first principles, ensuring that core rules are enforced consistently regardless of interface.

Presentation Separation:

The VMS integrates business logic directly into templates and views. This blurs the separation of concerns and risks duplicating logic across the presentation layer. A cleaner approach would involve serializers or service APIs to decouple presentation from business logic, improving reusability for other clients (mobile, APIs).

Strengths:

- Leverages Django's built-in admin for rapid CRUD.
- ORM simplifies relational data handling.
- Clear URL/view mapping supports straightforward navigation.

Weaknesses:

- Over-reliance on implicit ORM behavior for transactions.
- Business logic scattered across views/forms.
- Tight coupling between domain entities, reducing extensibility.

Recommendations:

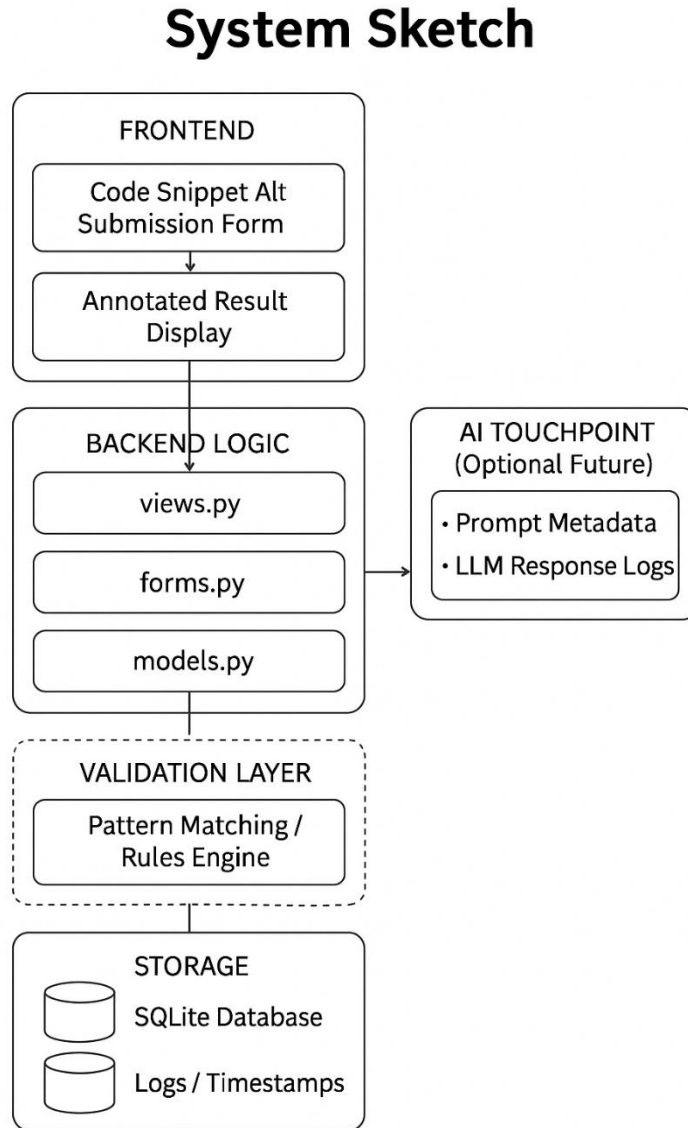
- Introduce explicit transaction management with `transaction.atomic`.
- Refactor business logic into models/managers or dedicated services.
- Decouple presentation with serializers/APIs.
- Adopt domain-driven design principles to better handle evolving workflows.

Conclusion:

The Django VMS architecture is functional and leverages Django's strengths in rapid development, but it falls short in enforcing invariants, transaction safety, and separation of concerns. By shifting toward service-oriented design and strengthening transaction boundaries, the system can achieve higher scalability, maintainability, and integrity.

System Sketch

Figure 1. Proposed architecture for hallucination-aware code validation system



References

1. NIST. *AI Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology, 2023. Available at: <https://www.nist.gov/itl/ai-risk-management-framework>

2. OWASP. *Top 10 for Large Language Model Applications*, 2023. Available at: <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
3. IEEE. *IEEE Standard 7001-2021: Transparency of Autonomous Systems*, IEEE Standards Association, 2021. Available at: <https://standards.ieee.org/ieee/7001/10231/>
4. W3C. *Web Content Accessibility Guidelines (WCAG) 2.2*, World Wide Web Consortium, 2023. Available at: <https://www.w3.org/WAI/WCAG22/>