National University of Computer and Emerging Sciences, Lahore

WardrobeWiz

Zoraiz Awan 21L-1835 BS(CS)

Taimoor Mohsin Roll No. 21L-5200 BS(CS)

Hassan Ali Murtaza Roll No. 21L-1799 BS(CS)

Supervisor: Muhammad Naveed

Final Year Project

November 21, 2025

Anti-Plagiarism Declaration

This is to declare that the work presented in this Final Year Project report, titled:

Title: WardrobeWiz

has been planned, implemented, and written by the undersigned authors. No part of this report has been copied or reproduced in its original form from any source without proper citation, and no section has been generated or inserted through direct copy–paste from external material. All referenced ideas, figures, and text have been appropriately acknowledged in the bibliography and used only to support our own arguments and design decisions.

We understand that plagiarism, whether intentional or unintentional, is a serious academic offence. In the event that this declaration is found to be false, we accept full responsibility for any academic or disciplinary consequences that may follow.

Date: .....

Name: Zoraiz Awan

Name: Taimoor Mohsin

Name: Hassan Ali Murtaza

Author's Declaration

We, the undersigned, hereby declare that the work presented in this report is the result of our own effort and has not been submitted, in whole or in part, to any other academic institution or organization for any degree, diploma, or qualification. Where material from other sources has been used, it has been properly cited and listed in the references section.

We further certify that the system implementation and results discussed in this report reflect the actual work carried out by the project team under the guidance of the supervisor mentioned on the title page.

Abstract

WardrobeWiz is an AI-driven wardrobe assistant that helps users make better use of the clothes they already own. Instead of recommending products from online stores, the system focuses on the user's personal wardrobe and suggests outfits that match their style, context, and preferences. Users upload photos of individual garments, which are processed by a vision model based on Contrastive Language–Image Pre-training (CLIP) to generate rich multimodal embeddings. These embeddings are stored in a vector database and later retrieved to assemble visually compatible and context-aware outfits.

The recommendation process is enhanced through retrieval-augmented generation, allowing the system to combine retrieved wardrobe items with lightweight natural language generation to explain why certain combinations work. A simple feedback loop—through actions such as like, dislike, or swap—lets WardrobeWiz gradually learn what each user prefers, leading to more personalised suggestions over time. The prototype is implemented using a modular ReactJS frontend and a FastAPI backend, integrated with FAISS for similarity search. Beyond convenience, the project aims to encourage more sustainable fashion behaviour by helping users rediscover underused garments, reduce impulsive purchases, and experiment with new styles using clothes they already own.

Executive Summary

Personal style decisions are made every day, yet many people still feel that they "have nothing to wear" despite having full wardrobes. Research reports by McKinsey and others indicate that a large proportion of clothing is worn only a handful of times, while purchases continue to grow year on year [1, 2]. Most digital fashion tools focus on shopping rather than the clothes people already own, and they rarely adapt to regional dressing styles or individual preferences. *WardrobeWiz* was conceived as a response to this gap: an AI-based wardrobe companion that learns from the user's existing clothing, provides outfit ideas for real-life contexts, and nudges them towards more mindful, sustainable use of their wardrobe.

The project combines three main strands of modern AI: computer vision to understand garments visually, retrieval-based search to find compatible items efficiently, and simple natural language generation to explain and justify outfits. Wardrobe photos are processed through a CLIP-based vision module and converted into dense embeddings, which are indexed in FAISS. When a user asks for an outfit—for example, "casual winter look for university"—WardrobeWiz retrieves suitable combinations from this index, reasons over style and context, and then presents a complete outfit with a short explanation of why the pieces work together.

From an engineering perspective, the system is built as a modular web application. A ReactJS frontend manages image upload, wardrobe browsing, and outfit display, while a FastAPI backend coordinates embedding generation, retrieval, and feedback handling. The current prototype focuses on a browser-based experience and a minimum viable product feature set: multi-image wardrobe ingestion, basic style profiling, outfit generation, and feedback logging. These choices keep the scope realistic for a Final Year Project while still demonstrating a full end-to-end AI pipeline.

The work is aligned with the United Nations Sustainable Development Goals, particularly SDG 9 (Industry, Innovation, and Infrastructure) and SDG 12 (Responsible Consumption and Production). By encouraging users to experiment with what they already own, WardrobeWiz reduces reliance on constant new purchases and showcases how AI can support more responsible consumption patterns in the fashion domain. The system also opens future opportunities for business models such as freemium styling services, white-label integration for fashion retailers, and analytics dashboards that summarise wardrobe usage trends at an aggregate level.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In recent years, artificial intelligence and computer vision have been integrated into many everyday applications, from photo search to content recommendation. Fashion has also benefited from this trend, particularly in areas such as product recommendation, size prediction, and virtual try-ons. However, most of these systems treat clothing purely as items to be bought rather than as part of the lived reality of a person's wardrobe. Many users continue to wear a small fraction of the garments they own, repeat familiar combinations, and feel overwhelmed by choice when getting ready. This Final Year Project investigates how AI can help at a more personal level by modelling a user's actual wardrobe and generating outfit ideas that respect their preferences, context, and cultural norms.

WardrobeWiz is an AI-powered wardrobe assistant that uses images of a user's own clothes as its core dataset. Each garment is embedded using a CLIP-based model and stored in a vector database. When the user requests an outfit, the system retrieves compatible pieces, composes them into a complete look, and accompanies the suggestion with a brief explanation. Over time, feedback signals such as likes, dislikes, or swaps allow the system to learn a personalised sense of style. The project sits at the intersection of computer vision, retrieval-augmented reasoning, and user-centred design, and aims to demonstrate how these strands can be combined into a usable, web-based prototype.

1.1 Purpose of this Document

The purpose of this report is to document the design, implementation, and evaluation of the WardrobeWiz system in a structured, academically rigorous way. It explains the motivation for the project, places it within the broader research and application landscape, and then describes how the system was engineered to meet the agreed goals. The document traces the project from the initial problem framing and literature review through to the software requirements, architectural design, and implementation details.

For each stage, the report makes explicit the assumptions, trade-offs, and constraints that shaped the final solution. The intention is that an academic examiner, supervisor, or future developer could read this report and understand not only what was built, but also why it was designed in this particular way. The document follows the structure prescribed by FAST-NUCES for Deliverable II: Chapter 1 introduces the project; Chapter 2 elaborates the vision and problem context; Chapter 3 reviews related research and applications; Chapter 4 presents the Software Requirements Specification (SRS); and Chapter 5 outlines the high- and low-level design.

1.2 Intended Audience

This report is primarily written for the Final Year Project evaluation committee, including the project supervisor, internal examiner, and external assessor. For this audience, the document aims to demonstrate that WardrobeWiz has been designed and implemented using sound software engineering practices, appropriate AI techniques, and a clear understanding of user needs. The level of technical detail in the SRS and design chapters is chosen to allow a technically literate reader to follow the architecture and reproduce or extend the system if required.

The report may also be useful for peers and junior students who wish to explore similar ideas in AI for fashion, digital wardrobes, or recommendation systems. While the focus is academic, the writing style is intended to remain readable for developers and practitioners who are comfortable with basic machine learning and web development concepts. Separate non-technical materials—such as a short pitch deck or demo video—can be prepared for potential industry partners or investors who are more interested in the value proposition than the implementation details.

1.3 Definitions, Acronyms, and Abbreviations

The following terms and abbreviations are used throughout this report:

- **AI**: Artificial Intelligence, the broad field concerned with building systems that can perform tasks that normally require human intelligence.
- **RAG**: Retrieval-Augmented Generation, an approach where external information is retrieved and passed into a language model to guide its responses.
- **CLIP**: Contrastive Language–Image Pre-training, a multimodal model that learns a shared embedding space for images and text.
- **RL**: Reinforcement Learning, a learning paradigm in which an agent receives rewards or penalties based on its actions and uses them to improve its policy.
- **FAISS**: Facebook AI Similarity Search, a library for efficient similarity search over dense vectors.
- **LLM**: Large Language Model, a neural network trained on large text corpora to generate and understand human language.
- **UX/UI**: User Experience / User Interface, the disciplines concerned with how users interact with and feel about software.
- **MVP**: Minimum Viable Product, the smallest feature set that demonstrates the core value of a

product.

- **SDG**: Sustainable Development Goal, part of the United Nations' 2030 agenda for global development.
- **API**: Application Programming Interface, a defined way for software components to communicate with each other.

1.4 Conclusion

This chapter has introduced the WardrobeWiz project, outlined the motivation behind it, and set expectations for what the reader will find in the rest of the report. The core idea is to shift AI in fashion away from pure shopping assistance and towards helping people use the clothes they already own in more creative and sustainable ways. By framing the work within the context of AI, personalisation, and responsible consumption, the chapter establishes why WardrobeWiz is both a technically interesting and socially relevant problem to tackle.

The following chapters build on this foundation. Chapter 2 develops the project vision in more detail, describing the problem domain, constraints, and stakeholders. Chapter 3 surveys academic and commercial work in related areas, showing how WardrobeWiz draws from and extends existing approaches. The later chapters convert this conceptual groundwork into concrete requirements, design artefacts, and implementation decisions.

Chapter 2 Project Vision

This chapter sets out the broader vision behind WardrobeWiz. It explains the problem space in which the system sits, the specific issues it is designed to address, and the goals and constraints that guided its development. The chapter also identifies the main stakeholders and describes how the project aligns with selected Sustainable Development Goals and potential future business opportunities.

2.1 Problem Domain Overview

WardrobeWiz is designed for everyday users who own more clothes than they comfortably manage. A typical wardrobe contains items bought at different times, for different purposes, and often in response to short-lived trends. Over time, many garments drift to the back of the closet and are rarely worn. Existing wardrobe apps largely treat clothes as static entries in an inventory, leaving users to do the work of combining pieces into outfits. In contrast, WardrobeWiz treats each wardrobe as a small, personalised dataset that can be analysed and recombined in intelligent ways.

At the heart of the problem domain is the challenge of translating clothing images into meaningful representations that support outfit generation. WardrobeWiz uses a CLIP-based vision module to embed each garment into a vector space that captures style-related similarities. These embeddings are stored in a FAISS index, which allows the system to quickly retrieve combinations of items that are visually or stylistically compatible. This technical core is wrapped in a user interface that makes it easy to upload images, request outfits for specific contexts, and provide feedback.

2.2 Problem Statement

Most digital fashion tools either encourage users to buy more clothing or offer only basic organisational features. They rarely start from the user's actual wardrobe, and they usually pay little attention to local dressing norms or non-Western clothing categories. As a result, users remain stuck in cycles of decision fatigue—feeling overwhelmed by choice—and wardrobe underutilisation, where a small fraction of items carry most of the wear.

The system therefore needs to address several intertwined problems. First, it should help users choose outfits more easily, by surfacing varied yet coherent combinations from their existing clothes. Second, it should personalise these suggestions to the user's tastes and lifestyle rather than relying solely on global fashion trends. Third, it should be flexible enough to accommodate cultural and regional dress forms, particularly for Pakistani and South Asian contexts where items like kurtas, shalwar kameez, and shawls play a central role. WardrobeWiz is proposed as an AI-based solution that combines visual

understanding, retrieval, and feedback to work within these constraints.

2.3 Problem Elaboration

A closer look at the problem domain reveals three main areas of concern. The first is decision fatigue. When a wardrobe contains dozens of usable combinations, choosing a single outfit each day becomes effortful. In practice, people fall back on safe, familiar looks. WardrobeWiz aims to offload some of this cognitive load by presenting a small set of curated suggestions that still feel varied, allowing users to say “yes” or “no” instead of starting from scratch.

The second area is the lack of deep personalisation and explainability in existing tools. Fashion recommendation models deployed in e-commerce settings typically learn from aggregated behavioural data and are optimised for clicks or purchases. They rarely explain why a particular item or outfit was suggested, and they do not adapt to subtle, long-term user preferences within a private wardrobe. WardrobeWiz addresses this by combining multimodal embeddings with simple, human-readable explanations, helping users understand why, for example, a particular blazer pairs well with a certain shirt and trousers.

The third area concerns cultural and regional fit. Much of the research literature on fashion recommendation is built on Western-centric datasets [3, 4]. While these are valuable starting points, they do not fully capture the diversity of garments, silhouettes, and layering styles seen in South Asia. WardrobeWiz is therefore designed to learn directly from the images that users upload, regardless of whether those items match typical Western categories, and can be extended later with regionally sourced datasets and brand imagery from local labels.

2.4 Goals and Objectives

The primary goal of WardrobeWiz is to build a practical, user-centred AI system that helps people get more value and enjoyment out of their existing clothes. This high-level aim can be unpacked into several concrete objectives. From a user perspective, the system should reduce the friction of choosing outfits, encourage experimentation with less-worn pieces, and support a feeling of control over one’s personal style. From a technical perspective, the project should demonstrate how CLIP-based embeddings, vector search, and simple language generation can be combined into a coherent, responsive web application.

Additional objectives include providing a clean, accessible interface for wardrobe ingestion and outfit browsing; integrating lightweight feedback mechanisms that allow the system to adapt over time; and structuring the codebase so that future extensions (such as mobile apps or fine-tuned models) can be

added without major architectural changes. Together, these objectives align the project both with Final Year Project expectations and with realistic pathways for further development.

2.5 Project Scope

For this Deliverable II and the current implementation phase, WardrobeWiz is scoped as a web-based Minimum Viable Product. The system supports user registration and login, wardrobe image upload, basic style profiling, outfit generation with textual explanations, and a simple feedback loop. All of these features are accessible through a browser-based ReactJS interface backed by a FastAPI server and FAISS-powered retrieval.

Several ideas are intentionally kept out of scope to keep the project manageable. These include building native mobile applications, implementing full e-commerce integration or "shop the look" functionality, and deploying large-scale fine-tuned language models. These features are discussed as future work, but the current focus remains on demonstrating the end-to-end pipeline of wardrobe ingestion, embedding, retrieval, and recommendation within a controlled, university-hosted environment.

2.6 Sustainable Development Goals

WardrobeWiz aligns most directly with SDG 9 (Industry, Innovation, and Infrastructure) and SDG 12 (Responsible Consumption and Production). By treating the personal wardrobe as a space for innovation, the system explores how AI infrastructure can be used to support more sustainable consumption patterns. Instead of encouraging constant new purchases, WardrobeWiz helps users rediscover and recombine existing items, potentially reducing the demand for fast fashion.

Figure 2.1: United Nations Sustainable Development Goal related to industry, innovation, and infrastructure.

In the long term, aggregated, anonymised wardrobe usage data could also offer insights into how people actually use their clothes, complementing the survey-based findings reported by McKinsey and oth-

ers [1]. Such insights could be valuable for both sustainability research and more responsible fashion industry planning.

2.7 Constraints

The project is subject to several practical constraints. Time is a major factor: the system must reach a demonstrable state within two academic semesters, alongside other coursework and commitments. Hardware limitations also shape the design. The team works with modest GPU access, so model choices favour lightweight, open-source architectures and efficient inference over very large or heavily fine-tuned networks.

Data availability is another constraint. There are limited public datasets that reflect South Asian clothing styles and layering patterns, so the system relies primarily on public fashion datasets and user-uploaded images. Finally, scope control is necessary to ensure that ambitious ideas—such as 3D try-on or full marketplace integration—do not dilute the core goal of building a robust wardrobe-focused recommender.

2.8 Business Opportunity

Although WardrobeWiz is developed primarily as an academic project, the underlying idea has potential commercial applications. A freemium model could offer basic wardrobe organisation and outfit suggestions for free, with paid tiers unlocking deeper analytics, seasonal capsule planning, and personalised style reports. Another possibility is to license the underlying API as a white-label service for fashion retailers, allowing them to integrate wardrobe-aware recommendations into their own apps.

Business-to-business partnerships could centre around "wardrobe-aware shopping", where a retailer suggests only items that complement what a customer already owns. Over time, anonymised wardrobe and usage statistics could be used to produce insights for brands that are serious about sustainability, such as understanding which categories are consistently underworn and adjusting production accordingly.

2.9 Stakeholders Description / User Characteristics

WardrobeWiz involves several stakeholder groups. The primary stakeholders are end users who upload their wardrobe images and rely on the system for outfit ideas. Their priorities include ease of use, trust in the recommendations, and a sense of style alignment. Secondary stakeholders include administrators and developers, who are responsible for maintaining the system, updating models, and ensuring data security. Potential future stakeholders include fashion retailers or sustainability-focused organisations interested in integration or analytics.

2.9.1 Stakeholders Summary

Stakeholder	Description and Role
End Users	Individuals who upload wardrobe images, receive outfit recommendations, and provide feedback.
Administrators	Manage the system, monitor performance, and handle user management and configuration.
Developers / Researchers	Extend the system, experiment with new models, and evaluate performance.
Retail Partners (Future)	Potential collaborators who may integrate WardrobeWiz as a service into their own platforms.

Table 2.1: Summary of WardrobeWiz Stakeholders

2.9.2 Key High-Level Goals and Problems of Stakeholders

End users are primarily looking for help making everyday style decisions without feeling overwhelmed. They want recommendations that reflect their taste and context, and they want to understand why a particular outfit is being suggested. Administrators and developers want a system that is stable, secure, and maintainable, with a clear separation between components so that individual modules can be updated or replaced. Future retail or research partners may be more interested in aggregated insights, integration options, and how the system can support their own sustainability or personalisation goals.

2.10 Conclusion

This chapter has expanded the initial idea introduced in Chapter 1 into a fuller project vision. It has described the problem domain, articulated the need for wardrobe-centred recommendations, and explained how WardrobeWiz aims to respond to both user needs and sustainability concerns. Constraints and business opportunities have also been considered to keep the project grounded in real-world conditions. The next chapter shifts focus to the academic and commercial landscape that surrounds WardrobeWiz, reviewing research and applications that inspired and informed the design.

Chapter 3 Literature Review / Related Work

This chapter reviews research and commercial systems related to WardrobeWiz. It surveys work on outfit recommendation, multimodal retrieval, and digital wardrobe tools, and highlights where current approaches fall short for the specific problem of helping users style their own clothes. The chapter also connects WardrobeWiz to broader trends in digital fashion and sustainability.

3.1 Definitions, Acronyms, and Abbreviations

List of Acronyms

Acronym	Definition
DNN	Deep Neural Network, a multi-layer model for learning complex patterns.
CNN	Convolutional Neural Network, a specialised DNN for image analysis.
RAG	Retrieval-Augmented Generation, a framework that combines retrieval and language generation.
CLIP	Contrastive Language–Image Pre-training, a model that learns aligned embeddings for images and text.
RLHF	Reinforcement Learning from Human Feedback, a technique for aligning models with user preferences.
MAE / MSE	Mean Absolute Error / Mean Squared Error, standard quantitative evaluation metrics.

Table 3.1: List of Acronyms and Their Definitions

3.2 AI in Fashion and Wardrobe Management

Artificial intelligence has become increasingly important in various fashion tasks, from product tagging and similarity search to trend forecasting. Early computer vision approaches for fashion relied on handcrafted features such as colour histograms and texture descriptors, which were limited in their ability to capture nuanced style relationships. Deep learning models, especially CNN-based architectures, improved classification and attribute prediction, but often treated garments in isolation rather than as parts of complete outfits.

Multimodal models such as CLIP further advanced the field by learning joint embeddings for images and text, enabling flexible task definitions that involve both visual and semantic information. Tangseng et al.

showed that outfit compatibility can be learned from data using deep neural networks, achieving high agreement with human judgements when grading outfits assembled from personal closets [3]. Deldjoo et al. provided a broad survey of deep learning techniques in fashion recommendation systems and organised them into a taxonomy that distinguishes between item-level and outfit-level recommenders, as well as different input modalities and feedback mechanisms [4]. Their work highlights persistent gaps in personalisation and explainability, which are central concerns for WardrobeWiz.

3.3 Outfit Recommendation Systems

Outfit recommendation systems attempt to move beyond single-item suggestions by assembling sets of garments that work together. Lin et al. introduced NOR, a Neural Outfit Recommendation model that jointly predicts outfit compatibility and generates natural-language comments describing the suggested look [5]. This dual output—prediction plus explanation—shows the potential of combining visual models with language generation to build more transparent recommendation systems.

Han et al. proposed FashionNet, which uses CNN-based feature embeddings to model compatibility relationships among clothing items [6]. Vasileva et al. focused on type-specific embeddings that better capture cross-category relationships, such as how particular tops combine with specific styles of bottoms or shoes [7]. Although these systems improved technical performance on benchmark datasets, they typically rely on curated catalogue images and do not directly model an individual's private wardrobe.

Bhardwaj et al. explored user-centric visual recommendations using crowd feedback, demonstrating that explicit user interactions can help refine model relevance and ranking [8]. Shen et al. extended this line of work with FashionRAG, a retrieval-augmented framework that uses CLIP embeddings and generative reasoning to produce outfit suggestions with textual rationales [9]. WardrobeWiz draws inspiration from these approaches but adapts them to a smaller, personal wardrobe setting with an emphasis on day-to-day usability rather than large-scale marketplace optimisation.

3.4 Visual Search and Multimodal Retrieval Systems

Visual search tools such as Google Lens and Pinterest Lens allow users to upload an image and retrieve visually similar items from large product inventories [10, 11]. These systems show how image embeddings and large-scale nearest neighbour search can support intuitive, content-based queries. However, they are designed primarily for product discovery and do not reason about the structure of an individual wardrobe or outfit-level compatibility.

Modern vector databases like FAISS provide efficient similarity search over high-dimensional embed-

dings and have become standard infrastructure for multimodal retrieval tasks. In fashion, they are often used to retrieve items that resemble a query image or meet certain attribute constraints. WardrobeWiz uses FAISS to index each garment in a user's wardrobe and to retrieve combinations that match a query embedding constructed from style preferences and contextual information. This retrieval is then combined with language generation to provide an explanation of the suggested outfit, making the system more transparent and engaging.

3.5 Digital Fashion and User Experience Platforms

Beyond academic research, several consumer-facing applications have attempted to digitise the wardrobe experience. Apps such as Smart Closet, Finery, and Whering allow users to catalogue their clothes, create outfit boards, and track what they wear over time [12, 13, 14]. These tools are useful for organisation, but they often require substantial manual input and rely on simple rule-based suggestions rather than adaptive AI.

At an industry level, reports by McKinsey and the World Economic Forum have highlighted both the environmental impact of fashion and the promise of digital tools to reduce waste [1, 15]. McKinsey notes that many garments are rarely worn, while digital fashion and AI stylists are positioned as possible ways to improve relevance and reduce returns. Adams' WardrobeSense project takes a more personal angle, analysing how little of an individual's closet is regularly used [16]. These findings reinforce the motivation for WardrobeWiz: there is both a sustainability argument and a user-experience argument for tools that help people make better use of their clothes.

3.6 Literature Review Summary Table

Study / System	Key Contribution	Relevance to WardrobeWiz
Tangseng et al. [3]	Outfit grading using deep neural networks.	Shows that outfit-level compatibility can be learned from personal closet data.
Deldjoo et al. [4]	Survey and taxonomy of fashion recommenders.	Identifies gaps in personalisation and explainability that WardrobeWiz addresses.
Lin et al. (NOR) [5]	Joint matching and comment generation for outfits.	Inspires the use of short textual rationales alongside visual recommendations.
Han et al. (FashionNet) [6]	Compatibility learning with deep visual features.	Provides a baseline approach for evaluating outfit compatibility.
Vasileva et al. [7]	Type-specific embeddings for fashion compatibility.	Supports the idea of modelling cross-category relationships within outfits.
Shen et al. (FashionRAG) [9]	Retrieval-augmented generation for outfit reasoning.	Demonstrates how CLIP and RAG can be combined, a core idea reused in WardrobeWiz.
Bhardwaj et al. [8]	Crowd-feedback-based visual recommendations.	Motivates the feedback loop used to adapt WardrobeWiz to individual preference.
Adams (WardrobeSense) [16]	Analysis of real wardrobe utilisation.	Provides evidence for underuse of garments and justifies wardrobe-focused tools.
McKinsey [1]	Industry survey on wardrobe behaviour.	Supports the sustainability and behaviour-change motivations for WardrobeWiz.
Statista [2]	Overconsumption statistics in fashion.	Reinforces the need for tools that encourage reuse rather than new purchases.
Smart Closet / Finery / Whering [12, 13, 14]	Digital wardrobe management apps.	Illustrate current practice and limitations in commercial wardrobe tools.
Pinterest Lens / Google Lens [10, 11]	Large-scale visual search engines.	Show the potential of image-based retrieval but lack wardrobe awareness.
World Economic Forum [15]	Analysis of digital fashion and AI stylists.	Places WardrobeWiz within broader digital fashion and sustainability trends.

3.7 Conclusion

The literature and applications reviewed in this chapter show that fashion recommendation has moved significantly beyond simple item similarity, with strong work on outfit compatibility, multimodal embeddings, and feedback-based personalisation. At the same time, there is still a gap when it comes to systems that work with a user's own wardrobe, respect local dressing norms, and actively encourage more sustainable use of clothing. WardrobeWiz is positioned within this gap. It builds on ideas from CLIP, retrieval-augmented generation, and user feedback, but applies them to the personal, everyday task of choosing outfits from what is already hanging in the closet.

Chapter 4 Software Requirement Specifications

This chapter specifies what WardrobeWiz is expected to do and the conditions under which it should operate. It describes the main features of the system, the functional and non-functional requirements, and the assumptions made about the environment in which the system will be used. The chapter also presents the key use cases, hardware and software requirements, the intended graphical user interface, database design, and a brief risk analysis. Together, these sections provide a clear technical baseline for implementation and later evaluation.

4.1 List of Features

WardrobeWiz offers a set of core features that together support the end-to-end experience of managing a digital wardrobe and generating outfits from it. The main features are listed below.

- User registration and login so that each person has a private, persistent wardrobe space linked to their account.
- Wardrobe ingestion, allowing users to upload images of garments and have basic metadata extracted through the AI vision pipeline.
- Style profiling through a short onboarding quiz and optional free-text prompts, used to build an internal style representation for each user.
- Outfit generation, where the system retrieves compatible items from the wardrobe index and presents them as a coherent look.
- Feedback handling so that users can like, dislike, or swap suggested outfits, allowing the system to learn their preferences.
- Outfit history and favourites, giving users a way to revisit previous suggestions that they found useful.
- An administration view for monitoring system health, basic usage patterns, and managing user accounts.

4.2 Functional Requirements

The functional requirements specify the behaviours that WardrobeWiz must support for different actors. The two main actor categories are general users and administrators. The requirements below are written at a level that can be directly traced to use cases and implementation tasks.

4.2.1 User Functions

- Users should be able to create an account using an email address or a supported identity provider (for example Google).
- The system should authenticate users securely and redirect them to their personal wardrobe dashboard after successful login.
- Users should be able to upload one or more images of garments, view the uploaded items, and delete items they no longer want in the system.
- After image upload, the system should generate embeddings and metadata for each garment and store these in the appropriate databases.
- Users should be able to request outfit suggestions from their wardrobe, optionally specifying context such as occasion or season.
- For each generated outfit, the system should display the selected garments and a short, human-readable explanation.
- Users should be able to provide feedback on an outfit in the form of like, dislike, or swap, and the system should record this feedback for later learning.
- Users should be able to mark outfits as favourites and access a simple history view of previous suggestions.
- Users should be able to update basic profile settings, such as style preferences or the answers to their initial style quiz.

4.2.2 Administrator Functions

- Administrators should be able to log in with elevated privileges that are distinct from general users.
- The system should provide a basic dashboard where administrators can view high-level statistics such as number of users, wardrobe size, and average response times.
- Administrators should be able to deactivate or remove user accounts when necessary.
- There should be a mechanism for administrators or developers to trigger model or embedding index updates without redeploying the entire system.
- The system should expose logs or diagnostic information to help administrators detect errors, performance bottlenecks, or unusual usage patterns.

4.3 Quality Attributes

In addition to core functionality, WardrobeWiz must meet several quality-related requirements so that it remains usable, reliable, and maintainable. The main quality attributes are summarised below.

1. Usability: The interface should be simple enough for non-technical users to navigate without training. Common actions such as uploading images or generating an outfit should require only a few clicks.
2. Reliability: Wardrobe data and embeddings should not be lost during normal operation. The system should behave predictably across repeated requests.
3. Scalability: The architecture should support gradual scaling, initially for a small group of pilot users but with the ability to grow to larger cohorts if needed.
4. Maintainability: The codebase should be modular and documented so that future students or developers can extend or refactor components without having to rewrite the entire system.
5. Security: User accounts and wardrobe images should be protected through secure authentication, appropriate access control, and encrypted communication channels.
6. Performance: From a user perspective, response times for generating an outfit should remain within a reasonable range, with a practical target of under ten seconds under typical load.

4.4 Non-Functional Requirements

Non-functional requirements provide more specific constraints and targets that support the quality attributes described above.

4.4.1 Reusability

The system should be implemented in a way that makes individual components reusable in other projects. For example, the CLIP embedding pipeline and FAISS retrieval code should be encapsulated so that they could be reused for a different recommendation task with minimal changes.

4.4.2 Performance

The system should be able to handle wardrobes of at least one thousand items per user without noticeable degradation in outfit generation time. Embedding generation may take longer during initial upload, but retrieval and generation for a request should remain responsive enough for interactive use.

4.4.3 Reliability

The system should handle transient failures—such as brief network interruptions or timeouts from the embedding model—gracefully, providing informative error messages and allowing users to retry operations. Automated tests should cover the most critical flows, including login, image upload, and outfit generation.

4.4.4 Security

All communication between the frontend and backend should use HTTPS. Passwords or tokens must not be stored in plain text. Administrator-only functions should be protected so that general users cannot accidentally access diagnostic or management screens.

4.4.5 Compatibility

WardrobeWiz should support recent versions of major browsers such as Chrome, Edge, and Firefox. The API layer should follow common REST conventions so that future mobile clients can be added without major redesign.

4.5 Assumptions

The requirements described above are based on several assumptions about the users and environment.

- Users have access to a stable internet connection and a device capable of running a modern web browser.
- Users can capture reasonably clear images of their garments, for example using a smartphone camera under normal indoor lighting.
- The university-hosted server or virtual machine has enough computational resources to run the CLIP model and FAISS index at the expected scale.
- Public fashion datasets and a small amount of custom data are sufficient to initialise the embedding and retrieval behaviour before user-specific learning takes over.
- The initial user base will be relatively small (for example, within the university community) during pilot testing.

4.6 Use Cases

The use cases in this section describe typical interactions between actors and the system. Each use case is summarised in a table with the main fields commonly used in software requirement documentation. In keeping with the university template, the tables are introduced by captions rather than additional subheadings.

Name	User Login
Actors	Registered user
Summary	The user logs in to access their wardrobe and outfit-generation functions.
Preconditions	User has a registered account.
Postconditions	User is authenticated and navigated to the dashboard.
Basic Flow	1. User opens login page. 2. User enters valid credentials. 3. System authenticates user and starts session. 4. User is redirected to dashboard.
Alternative Flow	Invalid credentials $\rightarrow$ error shown and retry allowed.

Table 4.1: Use Case 1 – User Login

Name	Wardrobe Upload
Actors	User
Summary	Uploads garment images for wardrobe ingestion.
Preconditions	User is logged in.
Postconditions	Images imported, processed, and embedded.
Basic Flow	1. User opens upload section. 2. User selects one or more images. 3. System uploads and processes each file. 4. Items appear in wardrobe view.
Alternative Flow	Unsupported file → system displays error.

Table 4.2: Use Case 2 – Wardrobe Upload

Name	Outfit Generation
Actors	User
Summary	System retrieves compatible garments and generates an outfit.
Preconditions	User has uploaded sufficient wardrobe items.
Postconditions	Outfit displayed with explanation.
Basic Flow	1. User opens outfit generation page. 2. Optional filters selected. 3. System retrieves compatible items. 4. Outfit and explanation displayed.
Alternative Flow	Insufficient wardrobe → system prompts for more items.

Table 4.3: Use Case 3 – Outfit Generation

Name	Feedback on Outfit
Actors	User
Summary	User provides feedback that updates personalization.
Preconditions	A generated outfit is visible.
Postconditions	Feedback stored and used by learning module.
Basic Flow	1. User views outfit. 2. User selects like/dislike/swap. 3. System stores feedback. 4. System updates preference model asynchronously.
Alternative Flow	Temporary backend issue → feedback queued.

Table 4.4: Use Case 4 – Feedback Interaction

Name	Admin Monitoring
Actors	Administrator
Summary	Admin reviews system metrics and activity.
Preconditions	Admin logged in with elevated privileges.
Postconditions	Metrics displayed and available for maintenance action.
Basic Flow	1. Admin opens dashboard. 2. System retrieves logs and metrics. 3. Admin reviews information.

Table 4.5: Use Case 5 – Admin Monitoring

4.7 Hardware and Software Requirements

The hardware and software requirements define the baseline environment assumed for development, deployment, and use.

4.7.1 Hardware Requirements

- A machine with a multi-core processor (for example, Intel Core i5 or equivalent).

- At least 8 GB of RAM to support the backend services and embedding computations.
- Around 20 GB of available storage for code, models, and stored images.
- A stable internet connection for both server and client interactions.

4.7.2 Software Requirements

- An operating system such as Windows 10, a recent Linux distribution, or macOS for development.
- Development tools including a code editor (for example Visual Studio Code) and Git for version control.
- Python 3.10 or later with FastAPI, Uvicorn, PyTorch, FAISS, and related libraries for the backend.
- Node.js and ReactJS for the frontend application.
- A relational database management system such as PostgreSQL or SQLite for metadata storage.
- A suitable environment for hosting the application, such as a university virtual machine or local server.

4.8 Graphical User Interface

The graphical user interface is designed to be minimal and functional, with a focus on clarity rather than visual complexity. A typical user journey begins on a login screen, followed by a dashboard that summarises the wardrobe and provides quick access to outfit generation.

The upload area allows users to drag and drop garment images or select files through a standard file chooser. Once images are uploaded, thumbnail views of garments appear in a grid layout. The outfit generation page displays one suggested look at a time, with images of each garment arranged in a simple layout and the accompanying explanation shown alongside or underneath. Feedback controls—such as like, dislike, and swap buttons—are placed near the outfit so that users can respond quickly without scrolling.

For administrators, a separate dashboard page presents lightweight charts or counters for overall wardrobe size, number of active users, and recent system activity. This separation ensures that normal users are not exposed to maintenance or diagnostic functionality.

4.9 Database Design

4.9.1 ER Diagram

The WardrobeWiz data model revolves around a small number of core entities. Users have accounts that link to their wardrobe items, which in turn are referenced by outfit records and feedback entries. An administrator entity is used to represent privileged accounts that can manage or observe the system more widely.

- The *User* entity stores login credentials and profile information such as style preferences.
- The *WardrobeItem* entity stores metadata about each garment, including links to its image and embedding.
- The *Outfit* entity represents combinations of garments generated by the system.
- The *Feedback* entity stores user responses to specific outfits.
- The *Admin* entity represents system administrators.

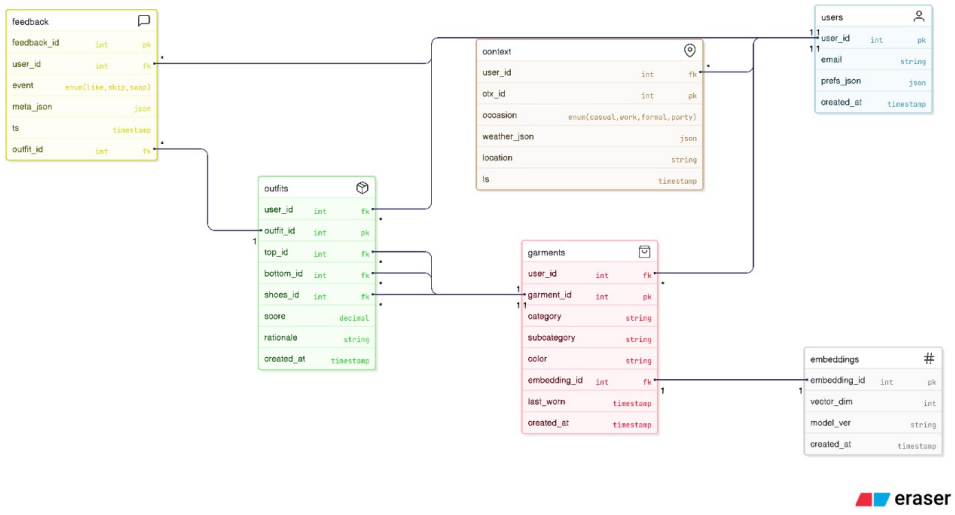

Figure 4.1: Entity–relationship diagram for WardrobeWiz.

4.9.2 Data Dictionary

Entity	Attribute	Type	Nullable	Description
User	user_id	Integer (PK)	No	Unique identifier for each user.
User	email	String	No	Email address used for login.
User	password	String	No	Securely hashed password.
User	style_vector	JSON	Yes	Encoded representation of stylistic preferences.
WardrobeItem	item_id	Integer (PK)	No	Unique identifier for each garment.
WardrobeItem	user_id	Integer (FK)	No	Links garment to its owner.
WardrobeItem	category	String	No	Type of garment (shirt, trousers, etc.).
WardrobeItem	color	String	Yes	Extracted or user-provided colour label.
WardrobeItem	embedding	Array(Float)	No	CLIP embedding vector representing the garment.
Outfit	outfit_id	Integer (PK)	No	Unique identifier for each generated outfit.
Outfit	user_id	Integer (FK)	No	Owner of the outfit.
Outfit	items	JSON	No	IDs of all garments included in the outfit.
Outfit	description	String	Yes	Explanation text generated by the system.
Feedback	feedback_id	Integer (PK)	No	Unique identifier for a feedback entry.
Feedback	user_id	Integer (FK)	No	User providing the feedback.
Feedback	outfit_id	Integer (FK)	No	Outfit on which feedback is given.
Feedback	rating	Integer	Yes	Numeric feedback (+1 like / –1 dislike).
Admin	admin_id	Integer (PK)	No	Unique identifier for admin.
Admin	email	String	No	Administrator's email.

Table 4.6: Data Dictionary for WardrobeWiz Entities

4.10 Risk Analysis

No system is free of risk, and WardrobeWiz is no exception. The table below summarises a few key risks identified during planning, along with their potential impact and high-level mitigation strategies.

Risk	**Impact**	**Mitigation Strategy**
Limited dataset diversity	Model may be biased toward Western clothing and misinterpret local apparel.	Gradually enrich training samples with user-uploaded images and small region-specific datasets.
Performance bottlenecks	Slow outfit generation could reduce user satisfaction and system usability.	Optimise embedding computations, cache repeated results, and profile slow sections of code.
User privacy concerns	Users may hesitate to upload clothing images due to security concerns.	Secure images using access controls, encryption, and transparent privacy policies.
Sparse feedback data	System may learn slowly if users rarely give explicit likes/dislikes.	Make feedback easy and unobtrusive to collect, and initialise with reasonable default preferences.
Infrastructure downtime	System may become inaccessible during demos or evaluation.	Implement logging, simple monitoring, and maintain VM snapshots for quick recovery.

Table 4.7: Risk Assessment and Mitigation Strategies for WardrobeWiz

4.11 Conclusion

This chapter has specified what WardrobeWiz is intended to do and the conditions under which it should operate. It has described the key features and functional requirements for both users and administrators, as well as the non-functional qualities and assumptions that shape the implementation. The use cases, hardware and software requirements, user interface overview, database design, and risk analysis all contribute to a clearer picture of the system from a requirements perspective. The next chapter builds on this foundation by describing the high-level and low-level design choices used to realise these requirements in software.

Chapter 5 High-Level and Low-Level Design

This chapter presents the architectural and detailed design of WardrobeWiz. It explains how the system's components interact, the principles guiding these decisions, and the strategies used to ensure scalability, maintainability, and user-centred performance. While the previous chapters described what the system must achieve, this chapter focuses on how those requirements are realised in practice. The design reflects a balance between technical feasibility, resource limitations within an academic environment, and the need to deliver a polished, intuitive user experience. Throughout this chapter, the system is broken down into subsystems, workflows, and class structures that collectively form the backbone of WardrobeWiz.

5.1 System Overview

WardrobeWiz follows a modular client–server architecture, where the frontend (React JS) interacts with a FastAPI backend through RESTful endpoints. This separation allows the user interface to remain lightweight and responsive while the backend manages compute-heavy tasks such as embedding generation and similarity search. The system processes user-uploaded images, generates embeddings using CLIP, stores them in a FAISS vector index, and retrieves compatible items during outfit generation. These retrieved results are passed through a lightweight reasoning module that produces human-readable explanations. The feedback mechanism closes the loop by incorporating user reactions into preference updates, gradually shaping more personalised recommendations.

In practice, the workflow is designed to feel seamless to the user. A wardrobe item uploaded on the frontend begins a chain of asynchronous tasks on the backend that prepare it for future outfit generation. When the user requests an outfit, the system only needs to perform retrieval and reasoning instead of reprocessing the wardrobe. The asynchronous handling of intensive operations keeps the interface responsive, even on modest academic hardware, while ensuring that the recommendation pipeline remains efficient for repeated use.

5.2 Design Considerations

The design of WardrobeWiz was shaped by both technical and practical constraints. Because the system is deployed on a university-provided virtual machine with limited compute power, the architecture prioritises lightweight models and efficient pipelines. This led to the use of CLIP as the embedding model, balancing accuracy with computational feasibility. Similarly, FAISS was selected for similarity search due to its suitability for CPU-based indexing and its ability to scale without excessive overhead.

Another key consideration was modularity. Each subsystem—Vision, Retrieval, RAG, and Feedback—was

intentionally separated so that future student groups or collaborators can extend or replace individual components without reworking the entire architecture. This modular design also supports experimentation; for instance, CLIP can later be swapped with a fine-tuned model, or the explanation generator can be replaced with a more advanced language model. Meanwhile, the frontend adheres to standard UI principles to ensure that students, researchers, or early adopters can navigate the system comfortably.

Security and reliability also influenced design choices. Authentication flows, error-handling structures, and careful management of user-uploaded images were included from the early design phase to prevent issues during evaluation or demonstrations. These considerations ensure that the system not only functions correctly but also meets the expectations of evaluators and end users.

5.3 System Architecture

The high-level architecture of WardrobeWiz consists of five interconnected modules:

- **Frontend (React JS):** Handles the user interface, wardrobe uploads, outfit display, and feedback interactions.
- **Backend Core (FastAPI):** Manages routing, authentication, integration between components, and asynchronous tasks.
- **Vision Module (CLIP):** Generates image embeddings and semantic labels from uploaded garments.
- **Retrieval and RAG Module:** Retrieves compatible items from FAISS and forms natural-language justifications.
- **Feedback Module:** Learns from user interactions and adjusts ranking preferences over time.

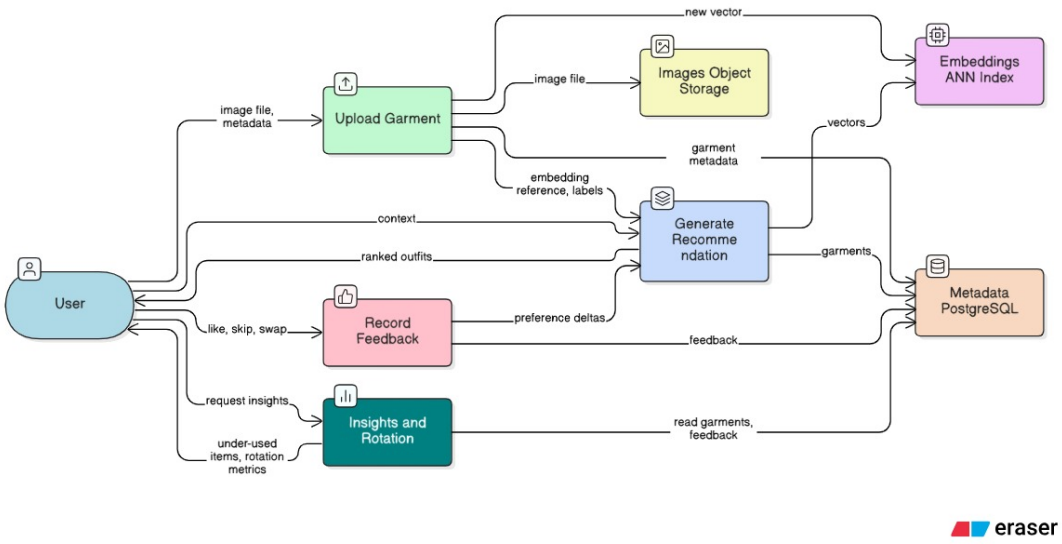

Figure 5.1: High-level system architecture of WardrobeWiz.

Figure 5.1 illustrates how these modules coordinate. The frontend initiates user requests, which flow into the backend where the Vision and Retrieval modules collaborate to analyse and assemble outfits. The Feedback module ensures that user behaviour influences future results, providing long-term personalisation.

5.4 Subsystem Architecture

Each subsystem encapsulates a distinct functionality while remaining loosely coupled with others. This improves maintainability and allows updates or replacements to occur without major structural changes.

5.4.1 User Subsystem

The user subsystem covers registration, login, wardrobe upload, and interaction features. It communicates with the backend through defined APIs and maintains local session data. It also provides client-side validation to prevent invalid uploads or incomplete forms. By keeping most operations asynchronous, the impact of heavy backend tasks is minimised for the user.

5.4.2 Admin Subsystem

Admins have access to a separate dashboard where system metrics, logs, and user activity summaries are displayed. Although minimal for the MVP, the admin subsystem is designed to accommodate future tools such as embedding index refresh, dataset management, or even fine-tuning triggers. All admin endpoints are protected through role-based access control.

5.4.3 Vision Subsystem

The Vision subsystem processes garment images through CLIP to produce embeddings and semantic labels. Embeddings are stored in FAISS for similarity search, while metadata is saved in PostgreSQL. Batch processing is supported to reduce overhead during bulk uploads. This subsystem forms the foundation of the entire recommendation pipeline.

5.4.4 Retrieval and RAG Subsystem

This subsystem handles outfit generation. The Retrieval component queries FAISS to find top-k compatible items based on similarity scores. The RAG component constructs prompts using retrieved items and generates short text explanations that describe the stylistic coherence of the outfit. This layered design ensures that the system produces both visually compatible and semantically meaningful recommendations.

5.4.5 Feedback Subsystem

The feedback subsystem plays a critical role in personalisation. User reactions act as signals that increase or decrease internal preference weights. Over time, the system learns which combinations the user prefers, making future suggestions more aligned with their style. By updating weights asynchronously, the system avoids introducing latency during normal outfit generation.

5.5 Architectural Strategies

The architectural strategies guiding WardrobeWiz focus on clarity, modularity, and long-term sustainability.

5.5.1 Programming Language Strategy

The project uses Python for backend components due to its strong ecosystem for machine learning and its compatibility with CLIP, FAISS, and FastAPI. React JS was selected for the frontend because of its reusable components, support for hooks, and ease of integration with REST APIs. This combination results in a clean and maintainable codebase that future groups can extend with minimal friction.

5.5.2 User Interface Paradigm

The interface follows well-established design principles such as consistency, clear feedback, and minimalism. Controls are placed where users expect them, and the layout avoids unnecessary clutter. Small

details such as clear thumbnails, meaningful loading indicators, and intuitive feedback buttons help create a more polished user experience.

5.5.3 Concurrency Strategy

To prevent blocking issues during heavy tasks like embedding generation, asynchronous programming is used throughout the backend. Functions that take longer—such as processing garments or updating weights—run in background threads. This ensures that the system stays responsive and that users never feel stuck waiting for the interface to respond.

5.5.4 Error Handling and Recovery

Error detection mechanisms catch backend issues such as failed model inference, missing files, or API exceptions. Logs are stored in structured format to simplify debugging. On the frontend, user-friendly messages inform the user when something goes wrong without revealing technical details.

5.5.5 Hosting and Version Control

The system is version-controlled through GitHub, following a branching model where new features are developed in isolated branches before being merged. The backend is deployed on a university virtual machine, served through Uvicorn and proxied with Nginx for reliability. This setup ensures predictable deployment and stable operation during evaluations.

5.6 Domain Model and Class Diagram

The domain model represents the core objects involved in WardrobeWiz. Users upload wardrobe items that have embeddings and metadata. These items are grouped into outfits, each of which may receive feedback. Admins oversee the system. Figure 5.2 shows the relationships between these classes and highlights how the system preserves object-oriented clarity.

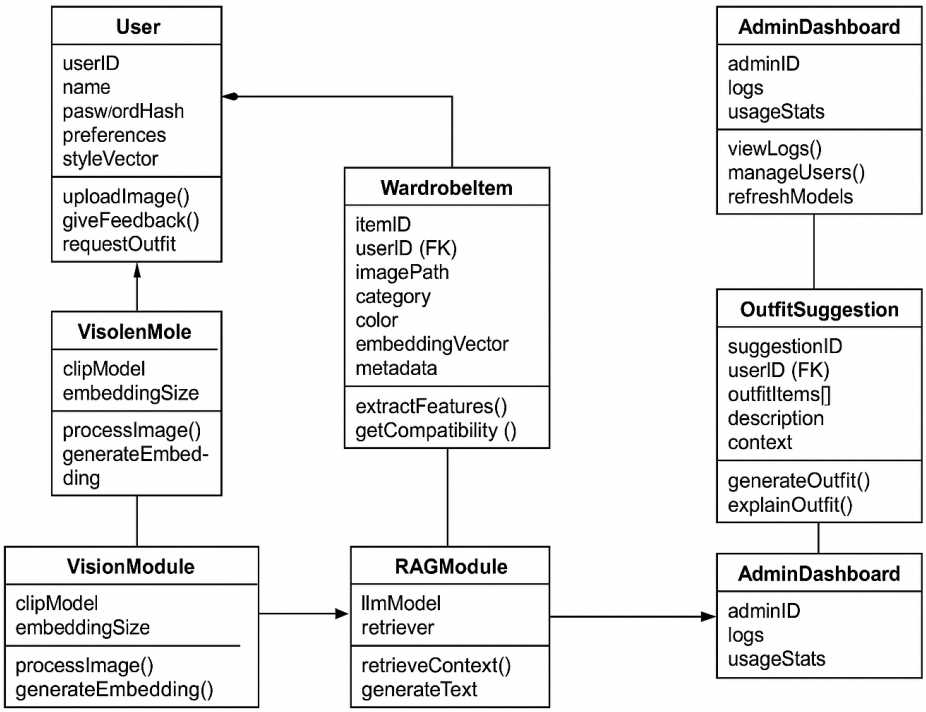

Figure 5.2: Class diagram for WardrobeWiz domain model.

The class structure is intentionally simple to keep the MVP manageable. However, the design leaves room for future expansion—such as garment attributes, multi-view image sets, or detailed event-based outfit histories. Careful separation between data models and business logic ensures that developers can refactor or extend the system without affecting core workflows.

5.7 Policies and Tactics

The development of WardrobeWiz follows several policies intended to ensure maintainability and code quality. Python code follows PEP 8 conventions, JavaScript code uses ESLint, and meaningful documentation is included both in code and in external notes. The hybrid database strategy—using PostgreSQL for metadata and FAISS for embeddings—supports efficient query behaviour without sacrificing relational structure.

Security tactics were included early, such as JWT-based authentication and environment-variable secrets. This reduces risks associated with poor credential handling. Engineering tactics include asynchronous pipelines, disciplined use of exceptions, and small, composable functions that maintain readability.

5.8 Conclusion

This chapter has outlined the design decisions, system architecture, and structural components that make up WardrobeWiz. By using a modular architecture, efficient embedding and retrieval pipelines, and a user-centred interface, the system provides both functional robustness and flexibility for future development. The next stage of the project involves refining the implementation, testing the interaction flows, and improving the personalisation loop. Overall, the design establishes a strong foundation for a scalable and sustainable AI-driven wardrobe recommendation platform.

Chapter 6 Implementation and Test Cases

This chapter describes the implementation work completed in FYP–1 for WardrobeWiz. The focus for this phase was to build a working prototype that demonstrates end–to–end wardrobe ingestion, image embedding, and basic storage for later retrieval. Rather than attempting to implement the entire recommendation pipeline in one step, the team agreed to first deliver a narrow but functional slice of the system: a user can upload a garment image through a web interface, the backend extracts a CLIP embedding, and the result is stored in a vector index together with basic metadata. This vertical slice validates core technical assumptions, such as model loading time, embedding latency, and feasibility of running the pipeline on a university virtual machine.

The implementation described in this chapter is intentionally modest but forms a concrete foundation for future work. It allowed the team to connect React, FastAPI, CLIP, FAISS, and PostgreSQL in a realistic setting instead of working with isolated code snippets. Test cases and formal evaluation for this prototype will be completed in FYP–2 once more features, such as outfit generation and feedback loops, have been implemented. For Deliverable III, the emphasis remains on explaining the design decisions, development environment, and practical challenges encountered while building this first prototype.

6.1 Implemented Prototype Overview

The WardrobeWiz prototype focuses on a single, coherent user journey:

1. The user logs in and opens a simple “Add Item” page.
2. The user uploads an image of a garment (for example, a shirt, trouser, or kurta).
3. The frontend sends the image to the backend through a REST API.
4. The backend loads a CLIP vision–language model, extracts an embedding, and predicts a coarse category label.
5. The system stores the embedding and metadata in a FAISS index and relational table.
6. The backend returns a response to the frontend so that the user can see a confirmation and basic information about the item.

This flow is deliberately limited to wardrobe ingestion and embedding rather than full outfit generation. However, it exercises the most critical integration points: image upload handling, model inference, vector indexing, and database persistence. Once this slice behaves reliably, it is straightforward to extend the same infrastructure to support outfit retrieval, ranking, and explanation in FYP–2.

6.2 Development Environment and Tools

The prototype was implemented using the same technology stack defined in earlier chapters, but in a simplified configuration to keep FYP–1 manageable. The backend runs on a university–hosted Ubuntu virtual machine with 8 GB RAM and 4 virtual CPUs, which is sufficient for running a CLIP model in inference mode. Python 3.10 and FastAPI were used to implement the REST API endpoints. Uvicorn serves the application during development, and PostgreSQL is used for structured data such as user accounts and metadata about wardrobe items.

On the frontend side, React JS was used to create a small, self–contained module for image upload and display. Axios was selected as the HTTP client library for sending multipart form–data requests to the backend. FAISS was installed on the backend to maintain an in–memory index of CLIP embeddings for each uploaded item. The implementation is kept minimal but mirrors the intended production architecture, which should reduce refactoring effort when more advanced features are developed in FYP–2.

6.3 Backend Implementation (FastAPI and CLIP Service)

The backend exposes a dedicated endpoint for ingesting wardrobe items. When the frontend sends a POST request with an image file, FastAPI validates the payload and saves the file temporarily in memory. The service then passes the image through a preprocessing pipeline, which resizes and normalises it to match the CLIP model's expected input format. The model used in the prototype is a publicly available CLIP variant, which returns a fixed–length embedding vector representing the visual features of the garment.

Once the embedding is computed, the backend constructs a record containing the user identifier, predicted category label, file path or object storage reference, and the embedding. The dense vector is inserted into a FAISS index, while the metadata is stored in PostgreSQL with a foreign key link to the user table. For debugging and transparency, the API returns a JSON response containing a short confirmation message, the predicted category, and the first few elements of the embedding vector so the team can verify that the pipeline is functioning. This design keeps the endpoint focused and simple, while still providing enough information to reason about performance and correctness.

6.4 Frontend Implementation (React Wardrobe Upload Component)

On the frontend, a dedicated React component was implemented to handle wardrobe uploads. The page presents a clean form with an image selector, a preview of the chosen file, and a submit button. When

the user selects an image, the component stores it in local state and displays a small thumbnail so the user can confirm they picked the correct file. This visual feedback proved helpful during informal trials, especially when multiple images were being uploaded in sequence.

When the form is submitted, the component builds a `FormData` object and sends it to the FastAPI endpoint using Axios. While the request is in progress, the button state changes to indicate loading, preventing accidental double submissions. On a successful response, the component displays a short summary of the result, including the predicted category and a simple "Item saved to wardrobe" confirmation. Errors such as missing files or backend connectivity issues are surfaced as user–friendly messages rather than raw stack traces. Although minimal, this implementation confirms that the frontend and backend can communicate reliably and that the image upload experience is intuitive for typical users.

6.5 Data Flow of the Prototype

The data flow in the prototype can be described as a linear pipeline that starts at the browser and ends in the database and vector index. The process begins when the user chooses an image on the React page. This file is wrapped in a multipart request and sent to the FastAPI server, where it is parsed and validated. The image then passes through the CLIP preprocessing and embedding functions, which convert it into a high–dimensional vector in a shared feature space.

After this transformation, the backend performs two parallel actions. First, it inserts the embedding into the FAISS index under a unique identifier. Second, it writes an entry into the PostgreSQL wardrobe table with fields such as item identifier, user identifier, category, and file reference. By separating the dense vector from the relational metadata, the design keeps both the similarity search and the database queries efficient. Finally, a compact response is returned to the frontend, closing the loop and allowing the user to move on to the next item. This pipeline lays the groundwork for future retrieval–based outfit generation in FYP–2.

6.6 Implementation Challenges and Resolutions

Although the prototype is small in scope, several practical issues surfaced during implementation. The first was model loading time. Initial attempts loaded the CLIP model on every request, which resulted in slow responses and unnecessary compute overhead. This was resolved by initialising the model once at application startup and keeping it in memory, reducing the per–request latency to an acceptable level for the prototype.

Another challenge involved handling image formats and sizes. Some test images were very large or

saved in uncommon formats, which caused occasional preprocessing errors. To address this, the backend now standardises input by converting images to RGB and resizing them to a fixed resolution before embedding. Finally, there were minor difficulties in aligning JSON responses between frontend and backend. These were solved by agreeing on a simple response schema early on and updating both sides consistently. Working through these issues gave the team a more realistic sense of the engineering work required beyond the theoretical design.

6.7 Summary of Implementation

In summary, the FYP–1 implementation delivers a working prototype that demonstrates the core ingestion and embedding pipeline for WardrobeWiz. Users can upload garment images through a React interface, the backend processes them using CLIP, and the resulting embeddings are stored in FAISS alongside structured metadata in PostgreSQL. Although the system does not yet generate full outfits or exploit reinforcement learning, this prototype validates the choice of tools and proves that the main components can be integrated on the available hardware.

Formal test cases, performance benchmarks, and user–centred evaluations are intentionally deferred to FYP–2, when more of the system will be in place and the behaviour of recommendations can be meaningfully assessed. For Deliverable III, the implemented slice is sufficient to demonstrate concrete progress and to justify the architectural decisions documented in earlier chapters.

Chapter 7 Conclusion and Future Work

This chapter summarises the work completed in FYP–1 for WardrobeWiz and outlines the planned activities for FYP–2. The first phase of the project focused on understanding the problem domain, studying related work in fashion recommendation and digital wardrobes, and defining a clear project vision aligned with sustainable consumption. The team then translated this vision into a detailed Software Requirements Specification and an architectural design that connects computer vision, retrieval–augmented generation, and feedback–based personalisation. Finally, a focused prototype was implemented to validate the core ingestion and embedding pipeline using React, FastAPI, CLIP, FAISS, and PostgreSQL.

Overall, FYP–1 has shifted the project from an abstract idea to a concrete, technically grounded system. The design and implementation work completed so far provide a realistic basis for scaling WardrobeWiz into a full wardrobe recommendation platform in FYP–2. At the same time, the team has gained a clearer understanding of the constraints imposed by hardware, data availability, and development time, which will influence how features are prioritised in the next phase.

7.1 Reflection on Prototype and Design

The prototype delivered in FYP–1 may appear small compared to the full ambition of WardrobeWiz, but it plays a critical role in de–risking the project. By implementing wardrobe ingestion and CLIP–based embedding, the team has demonstrated that the chosen stack can run on the available infrastructure and that the basic data flow is viable. This is preferable to attempting to design a complex system entirely on paper and then discovering late in FYP–2 that key assumptions do not hold.

The architectural and design work produced in earlier chapters has also been helpful in maintaining a coherent direction. The separation of concerns between frontend, backend, vision, retrieval, and feedback modules will make it easier to grow the system incrementally. However, the current implementation still lacks many user–facing features and does not yet generate recommendations, so the project cannot be considered complete from an end–user perspective. FYP–2 will therefore need to focus on turning this technical foundation into a truly usable application.

7.2 Limitations of Current Work

The main limitation of the current system is its restricted functionality. WardrobeWiz can ingest and embed garments, but it does not yet generate outfits, rank combinations, or adapt to user preferences through reinforcement learning. As a result, the prototype does not yet showcase the full value proposition of reducing decision fatigue and promoting sustainable wardrobe use. The absence of user–facing

analytical features, such as wardrobe utilisation statistics or favourite outfits, also limits the perceived usefulness of the system at this stage.

From a technical standpoint, the current implementation is limited by a small internal dataset and a lack of real user trials. Embeddings are computed for a small number of garments uploaded by the team, and the performance of CLIP on culturally specific garments, such as traditional South Asian clothing, has not been thoroughly evaluated. In addition, there is no formal testing framework in place yet. Automated unit, integration, and performance tests will be essential in FYP–2 to ensure that the system remains stable as more features are added.

7.3 Future Work and FYP–2 Plan

The second phase of the project will focus on transforming WardrobeWiz from a technical prototype into a functional wardrobe assistant that delivers meaningful recommendations. The planned work for FYP–2 can be organised into several themes. The first priority is to implement the full recommendation loop. This includes retrieving compatible garments from the FAISS index, assembling complete outfits based on context, and generating natural–language explanations that describe why particular items work well together. Lightweight feedback mechanisms such as like, dislike, and swap actions will be wired into a simple reinforcement learning layer so that recommendations improve as the user interacts with the system.

A second stream of work will concentrate on user experience and interface design. The current upload page will be expanded into a richer dashboard that displays the wardrobe, recommended outfits, favourites, and basic statistics about item usage. A short style–profiling questionnaire will be integrated to initialise user preferences. At the same time, basic mobile responsiveness will be added so that users can access WardrobeWiz comfortably from different devices during everyday use.

A third theme concerns evaluation and robustness. FYP–2 will introduce structured test cases for each major module, covering both functional correctness and performance under typical load. Small–scale user testing sessions will be conducted with volunteers to gather qualitative feedback on recommendation quality, interface clarity, and perceived usefulness. The team will also explore augmenting the dataset with additional garments—especially culturally diverse items—to reduce bias and improve the relevance of outfit suggestions in a South Asian context.

Finally, the project will allocate time towards polishing and documentation near the end of FYP–2. This includes code refactoring where required, improving error handling and logging, and preparing deployment scripts that make it easier to demonstrate the system on different machines. A refined

final report and presentation will summarise the complete journey from concept to implementation and evaluation, highlighting how WardrobeWiz contributes to more thoughtful and sustainable wardrobe usage.

Bibliography

[1] McKinsey & Company, "Consumers are reconsidering what's in their closets – the state of fashion 2021," 2021. [Online]. Available: https://www.mckinsey.com/industries/retail/our-insights/consumers-are-reconsidering-whats-in-their-closets-the-state-of-fashion-in-2021 [Accessed: Oct. 10, 2025].

[2] Statista, "Consumers buy more clothes than they need – overconsumption statistics," 2025. [Online]. Available: https://www.statista.com/chart/31299/consumers-buy-more-clothes-than-they-need/ [Accessed: Oct. 10, 2025].

[3] P. Tangseng, T. Okatani, and T. Yamasaki, "Outfit grading and recommendation from personal closets using deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1–8, 2017. [Online]. Available: https://arxiv.org/abs/1804.09979 [Accessed: Oct. 10, 2025].

[4] Y. Deldjoo, T. D. Noia, M. F. Dacrema, and P. Cremonesi, "A survey on deep learning techniques for fashion recommendation systems," *ACM Computing Surveys*, vol. 55, no. 3, pp. 1–40, 2022.

[5] X. Lin, Z. Chen, and Y. He, "Explainable outfit recommendation with joint matching and comment generation (nor)," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*, pp. 1053–1062, 2019.

[6] X. Han, Z. Wu, Y. Jiang, and Q. Jin, "Fashionnet: Learning outfit compatibility with deep feature embeddings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3217–3226, 2020.

[7] M. Vasileva, B. Plummer, E. Mettler, and A. Zisserman, "Learning type-specific embeddings for fashion compatibility," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 390–405, 2018.

[8] A. Bhardwaj, S. Garg, and R. Sharma, "User-centric visual recommendations using crowd feed-

back for improved fashion personalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2020)*, pp. 712–720, 2020.

[9] J. Shen, H. Lee, and J. Park, "Fashionrag: Retrieval-augmented generation for fashion outfit reasoning," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '23)*, vol. 37, pp. 5834–5841, 2023.

[10] Pinterest, "Pinterest lens," 2023. [Online]. Available: https://www.pinterest.com/lens/ [Accessed: Oct. 10, 2025].

[11] Google, "Google lens," 2023. [Online]. Available: https://lens.google.com/ [Accessed: Oct. 10, 2025].

[12] Smart Closet, "Smart closet app," 2023. [Online]. Available: https://www.smartcloset.com [Accessed: Oct. 10, 2025].

[13] Finery, "Finery app," 2023. [Online]. Available: https://www.finery.com [Accessed: Oct. 10, 2025].

[14] Whering, "Whering app," 2023. [Online]. Available: https://www.whering.com [Accessed: Oct. 10, 2025].

[15] World Economic Forum, "Digital fashion: Where technology, culture and creativity collide," 2025. [Online]. Available: https://www.weforum.org/stories/2025/06/digital-fashion-where-technology-culture-and-creativity-collide/ [Accessed: Oct. 10, 2025].

[16] H. Adams, "Wardrobesense: How much of your closet do you really wear?," 2023. [Online]. Available: https://www.hannaadamss.com/wardrobesense [Accessed: Oct. 10, 2025].