GITGOOD

Help gitGood, gitBetter

ARCHITECTURAL SPECIFICATIONS

DEMO 1

WEATHER TO WEAR

EPI-USE®

KYLE LIEBENBERG DIYA BUDHIA ALISHA PERUMAL	IBRAHIM SAID TAYLOR SERGEL

Table of Contents

INTRODUCTION

Weather to Wear is a mobile application that aims to simplify weather forecasts into a personalized wardrobe consultant. By analyzing real-time weather forecasts with the combination of a user's personalized styling preferences, it uses AI-driven outfit suggestions to provide appropriate fashion recommendations from a user's personal clothing collection.

Architectural specifications for Weather to Wear are important because they provide a structured outline to ensure the system meets its functional and quality requirements. These will guide the development of the systems features such as the AI-driven outfit recommendations, clothing catalog, social platform, the virtual closet and emergency responses, and the Fashion Time Machine. By specifying standards for caching, on-device AI training, and device compatibility, architectural requirements ensure a robust, user-friendly PWA that minimizes costs, supports growth, and maintains user trust through secure and reliable operation. Without these specifications, the project risks misalignment, performance issues, or budget overruns, compromising its ability to deliver a seamless, weather-appropriate styling experience. Architectural and design decisions were made by the team based on the quality requirements of the system which were decided in discussion with the client. The other strategies including decomposition and generating test cases will be employed during the implementation of the system to allow for parallel development and ensuring the system meets all the functional requirements. However the main strategy employed during the architectural design phase was designing based on the quality requirements of the system.

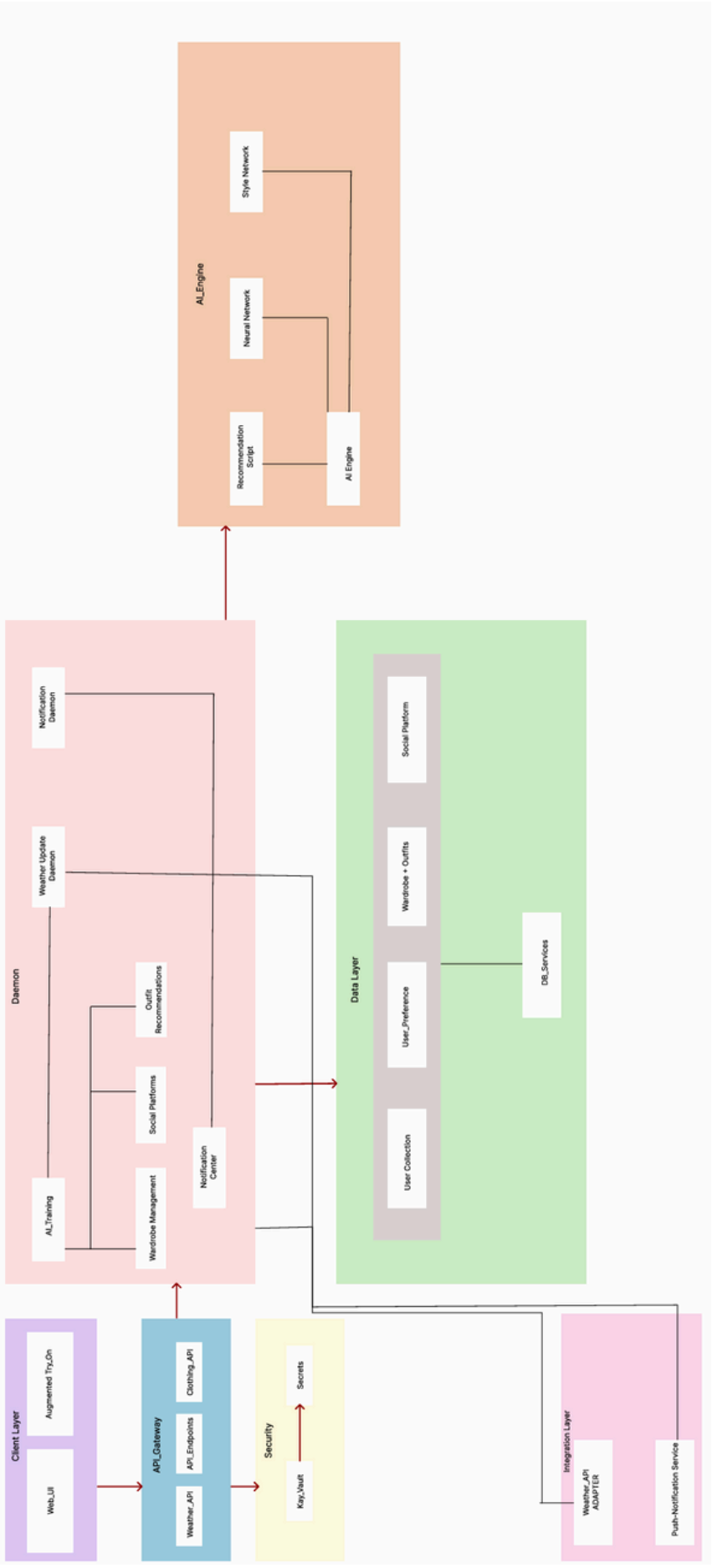
The architectural specifications and requirements for Weather to Wear focus on ensuring reliability, extensibility, performance, availability, scalability and security for the Progressive Web App (PWA).

QUALITY REQUIREMENTS

The quality requirements for Weather to Wear are embedded within the architectural and delivery requirements, focusing on ensuring a reliable, performant, accessible, and secure Progressive Web App (PWA) that meets user expectations. Provided below is a summary of the quality requirements, to ensure delivering AI-driven outfit recommendations, weather API integration, social features, and the Fashion Time Machine feature:

1. RELIABILITY
   - The system must maintain consistent functionality by mitigating reliance on external APIs through redundancy, by using multiple APIs, and caching, ensuring uninterrupted access to weather data and outfit recommendations.
2. PERFORMANCE
   - Outfit generation must occur within reasonable timeframes to provide quick, seamless recommendations, critical for user satisfaction and the AI recommendation engine.
   - Image processing must be optimized for efficient performance, minimizing load times and resource usage.
3. AVAILABILITY
   - The PWA must operate on common devices with graceful fallback options for older technology to ensure broad accessibility.
   - Offline caching must enable functionality with intermittent internet connectivity, supporting usability in varied network conditions.
4. SCALABILITY
   - Minimize external API calls through caching and efficient data management to reduce costs and support a growing user base.
   - Prioritize on-device training for the AI recommendation engine to lower server compute costs, enhancing scalability.
5. SECURITY
   - User data must be properly partitioned to ensure privacy and compliance with data protection standards.
   - Client-server communication must use industry-standard TLS encryption to secure data transmission, fostering user trust, especially for social platform interactions.
6. ACCESSIBILITY
   - The PWA must minimize mobile data usage and optimize battery consumption during extended use to enhance user experience on resource-constrained devices.

ARCHITECTURAL DIAGRAM



ARCHITECTURAL PATTERNS

LAYERED PATTERN (N-TIER ARCHITECTURE)

The Layered architectural pattern organizes the system into horizontal layers, with each layer interacting with the one below it, promoting separation of concerns and modularity.
The architecture diagram is divided into distinct layers: Client Layer, API Gateway, Daemon, Data Layer, Integration Layer, and elements like Security and AI Engine; with each layer having specific responsibilities.

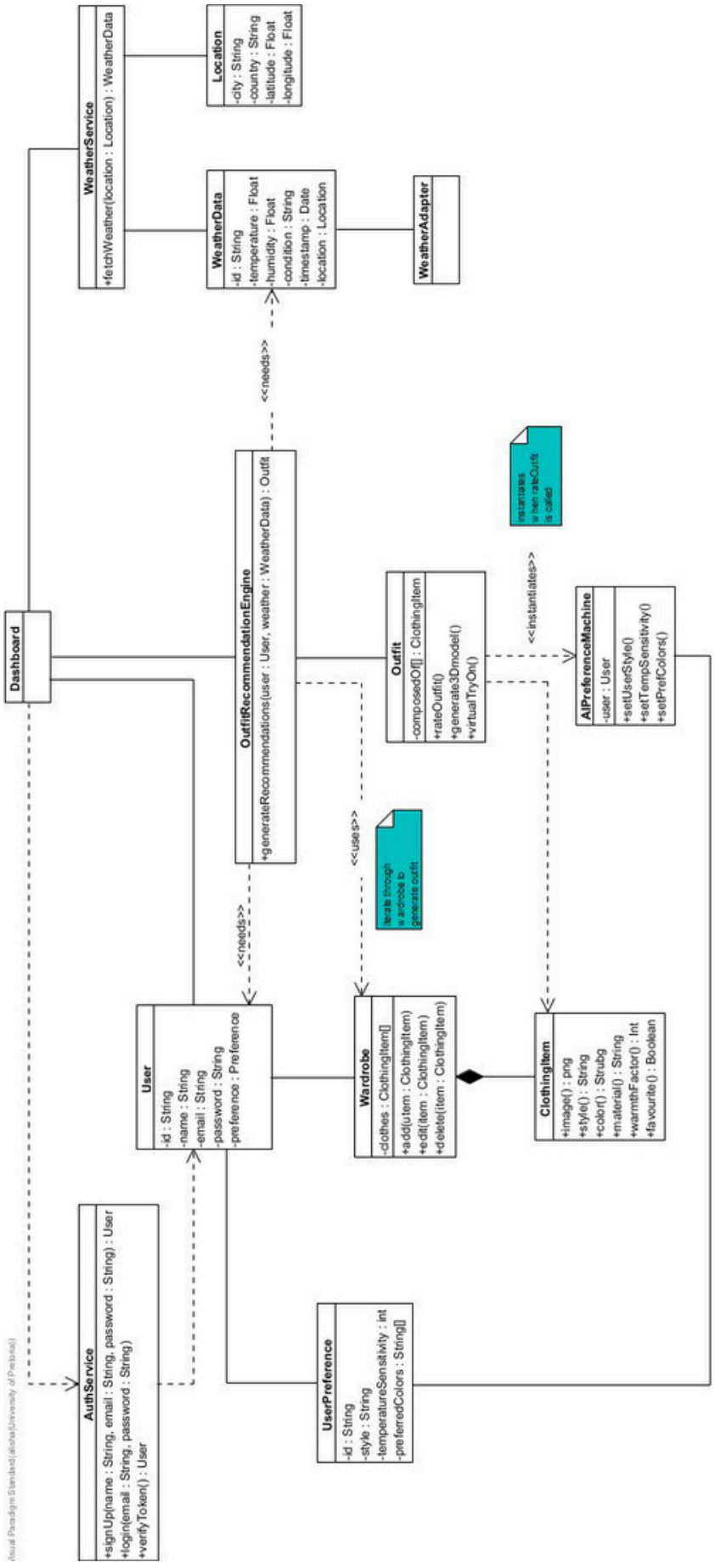
MICROSERVICES PATTERN

The system is broken into small, autonomous services that can be developed, deployed, and scaled independently, suggesting a microservices architecture.
The architectural diagram shows multiple independent services within the Daemon layer and the AI Engine. These services handle specific functionalities and communicates via the API Gateway or directly with the Data Layer.

EVENT-DRIVEN PATTERN

The system uses events to trigger actions across components, enabling loose coupling and real-time responsiveness.
The daemons and the Notification Center suggest asynchronous processing. These components likely respond to events, such as weather updates or user actions, triggering notifications or recommendations.

CLASS DIAGRAM



DESIGN PATTERNS

FACADE DESIGN PATTERN

The Facade Design Pattern can be found in the interaction between the Dashboard and each subsystem in the class diagram (AuthService class, WeatherService class, OutfitRecommendationEngine class, User class and Wardrobe class).
Purpose: The Facade pattern ,through the Dashboard provides a single entry point for the user interface to interact with multiple components like weather fetching, outfit generation, and authentication.

SINGLETON DESIGN PATTERN

The Singleton Design Pattern can be found in the AuthService class, where centralized authentication logic can be found. There exists only one shared instance that handles all login/signup/verification operations.
Purpose: The Singleton pattern ensures that the AuthService class has only one instance and provides a global point of access to that instance.

ADAPTER DESIGN PATTERN

The Adapter Design Pattern can be found in the WeatherAdapter class, where data from the weather APIs are adapted into WeatherData objects.
Purpose: The Adapter Pattern is used to convert the interface of a class into another interface that clients expect.

FACTORY DESIGN PATTERN

The Factory Design Pattern can be found in the Outfit class, that is involved in the creation of Outfits made up of ClothingItems.
Purpose: The Factory Pattern helps centralize and abstract the creation of Outfit objects based on user and weather data. Factory Method is used in the OutfitRecommendationEngine class to encapsulate the creation of Outfit objects.

COMPOSITE DESIGN PATTERN

The Composite Design Pattern allows individual clothing items and groups of items to be treated uniformly as a single entity.
Purpose: The Composite Pattern enables the system to treat individual clothing items and groups of items (Outfits) the same way.