High Level Design

MisMatch

version 1.3
30.11.2025

Document history

Version	Status	Date	Responsible person	Reason for change
1.0	Draft	26.11.2025	Rana Soliman	Added activity diagrams
1.1	Draft	27.11.2025	Sabhyeta Khadka, Nancy Mroke	Purpose and Sequence diagrams created
1.2	Draft	28.11.2025	Fabio Jindrak	Class Diagram added
1.3	Final	30.11.2025	Fabio Jindrak, Sabhyeta Khadka, Nancy Mroke, Rana Soliman	Review and final touches

Table of contents

- 2 Punkte
  - Zweck der App, Systemüberblick, Definitionen wenn notwendig
  - Komponenten des Systems: Rolle und Funktionalität pro Komponente
  - Architektur: Wie bilden die Komponenten das Gesamtsystem

1. Introduction

1.1. Purpose

MisMatch is a web-based virtual wardrobe application designed to eliminate daily outfit decision fatigue by providing users with a simple, intuitive platform to visualize and create clothing combinations. The application addresses the common problem of spending excessive time choosing outfits by offering:

- **Digital Outfit Visualization**: A 2D interface that allows users to see complete outfit combinations before wearing them
- **Quick Mix-and-Match Functionality:** Manual browsing through clothing items with simple navigation controls
- **Random Outfit Generation:** An automated "MisMatch" feature that creates unexpected yet wearable outfit combinations
- **Outfit Management:** Save, edit, and retrieve favorite outfit combinations for future reference
- **Privacy-Focused Solution:** A personal wardrobe tool without social features or data sharing

The core value proposition is simplicity and efficiency: helping users maximize their existing wardrobe without the complexity of shopping integrations, AI recommendations, or social networking features.

1.2. System Overview

MisMatch is built as a three-tier web application following the Model-View-Controller (MVC) architectural pattern:

Presentation Tier: HTML/CSS/JavaScript –> User Interface
Application Tier: Python Web Framework –> Buisness Logic
Data Tier: MongoDB -> User Data and Clothing Inventory

The system operates as so:

1. User Interaction: Users access the application through a web browser
2. Request Processing: Python web framework backend processes user requests and executes business logic
3. Data Management: MongoDB stores user accounts, outfit configurations, and clothing metadata
4. Response Delivery: Rendered HTML pages with clothing visualizations are returned to the browser

1.3. Definitions

TERM	DEFINITION
HLD	High-Level Design: architectural overview of the system
SRS	Software Requirements Specification
UI	User Interface: the visual elements users interact with
API	Application Programming Interface
MVC	Model-View-Controller: software architectural pattern
REST	Representational State Transfer: web service architectural style
Session	Server-side storage of user state during a browsing session
MongoDB	NoSQL document-oriented database system
DAO	Data-Access-Object

2. System Components

System is built up of following components:

1. Presentation Layer Components
2. Application Layer Components
3. Business Logic Layer Components
4. Data Access Layer Components
5. Data Layer Components

2.1. Presentation Layer Components

2.1.1. HTML Templates

Role: Dynamic page generation with server-side rendering

Functionality:

- Login/Sign-up forms
- Main wardrobe interface with clothing display boxes
- Saved outfits gallery view
- User settings/profile pages
- Error and success message displays

Technology: Server-side templating engine (integrated with Python web framework)

2.2. Application Layer Components

2.2.1. Authentication Controller

Role: Manages user registration, login, and logout operations

Functionality:

- **User Registration**:
    - Validates new user input (username, password)
    - Checks for duplicate usernames
    - Hashes passwords using a secure password hashing algorithm
    - Creates new user record in database
- **User Login**:
    - Authenticates credentials against database
    - Establishes secure session upon successful login
    - Handles login failures with appropriate error messages
- **User Logout**:
    - Terminates user session
    - Redirects to login page

Interfaces:

- **Input:** Login credentials, registration data
- **Output:** Session tokens, authentication status
- **Database:** Interacts with users collection

2.2.2. Outfit Generator Controller

Role: Handles outfit creation requests, both manual and random

Functionality:

- **Manual Selection**: Processes user selection of individual clothing items
- **Random Generation** ("MisMatch" feature):
    - Randomly selects one item from each category (tops, bottoms, footwear)
    - Ensures valid combinations (no duplicates within same category)
    - Returns complete outfit configuration
- **Outfit Display**: Coordinates display of selected outfit on interface

Interfaces:

- **Input:** User selections or random generation request
- **Output:** Outfit configuration (3 clothing item IDs)
- **Database:** Queries clothing collection for available items

2.2.3. User Session Manager

Role: Maintains user state across multiple HTTP requests

Functionality:

- Stores user ID in session after login
- Validates session on each request to protected routes
- Implements session timeout handling
- Manages session security (secure cookies, CSRF protection)

Technology: Session management with secure cookie storage

Session Data Stored:

- User ID
- Username
- Login timestamp
- Current outfit state (temporary, not persisted)

2.2.4. Wardrobe Item Controller

Role: Manages browsing and filtering of clothing items

Functionality:

- **Navigation**: Implements arrow-based item browsing
  - Left arrow: Show previous item in category
  - Right arrow: Show next item in category
  - Circular navigation (wraps around at list ends)
- **Filtering** (Should-have feature):
  - Filter by category (tops, bottoms, footwear)
  - Filter by color (Could-have feature)
- **Image Retrieval**: Fetches clothing item images from database or file system

Interfaces:

- **Input:** Navigation commands, filter criteria
- **Output:** Clothing item data with image paths
- **Database:** Queries clothing collection

2.2.5. Outfit Manager Controller

Role: Handles saved outfit operations

Functionality:

- **Save Outfit**:
  - Generates default outfit name (e.g., "Outfit 1", "Outfit 2")
  - Stores outfit configuration linked to user ID
  - Returns confirmation message
- **Retrieve Outfits**:
  - Fetches all saved outfits for logged-in user
  - Sorts outfits by creation date (optional: user-defined sorting)
- **Edit Outfit**:
  - Loads existing outfit configuration
  - Allows modification of clothing items
  - Updates outfit record in database
- **Rename Outfit** (Should-have):
  - Updates outfit name based on user input
- **Delete Outfit**:
  - Removes outfit record from database
  - Confirms deletion to user

Interfaces:

- **Input:** Outfit data, user commands
- **Output:** Outfit records, operation status
- **Database:** Save, retrieve, update, and delete operations on outfits collection

2.3. Business Logic Layer Components

2.3.1. User Management Logic

Role: Core business logic for user account operations

Functionality:

- Validates registration data (password strength)
- Enforces unique username
- Implements password hashing using bcrypt or similar
- Manages user profile updates (Could-have: username/password changes)

Security Considerations:

- Password minimum length: 8 characters
- Hash algorithm: bcrypt with appropriate salt rounds
- No plain-text password storage

2.3.2. Outfit Creation Service

Role: Business rules for outfit composition

Functionality:

- Validates outfit completeness (must have top, bottom, footwear)
- Enforces outfit structure rules
- Generates outfit previews for display
- Calculates outfit metadata (creation date, last modified)

Business Rules:

- Each outfit must contain exactly 3 items (one from each category)
- No duplicate items within a single outfit
- All items must exist in clothing database

2.3.3. Random Outfit Generator Service

Role: Algorithmic generation of random outfit combinations

Functionality:

- Selects random items from each clothing category
- Ensures variety (avoids recently generated combinations)
- Implements pseudo-random selection with seeding
- Returns valid outfit configuration

Algorithm Approach:

1. Query available tops, bottoms, footwear from database
2. Use random.choice() to select one item from each category
3. Validate selection doesn't duplicate current outfit
4. Return outfit configuration

2.4. Data Access Layer Components

2.4.1. User DAO (Data Access Object)

Role: Abstracts database operations for user data

Functionality:

- create_user(username, hashed_password): Insert new user
- get_user_by_username(username): Retrieve user for login
- get_user_by_id(user_id): Fetch user details
- update_user(user_id, updates): Modify user profile
- delete_user(user_id): Remove user account

Database Interactions: Direct MongoDB queries on users collection

2.4.2. Outfit DAO

Role: Abstracts database operations for outfit data

Functionality:

- create_outfit(user_id, outfit_name, clothing_ids): Save new outfit
- get_outfits_by_user(user_id): Retrieve all outfits for user
- get_outfit_by_id(outfit_id): Fetch specific outfit
- update_outfit(outfit_id, updates): Modify outfit (rename, change items)
- delete_outfit(outfit_id): Remove outfit

Database Interactions: CRUD operations on outfits collection

2.4.3. Clothing DAO

Role: Abstracts database operations for clothing item data

Functionality:

- get_all_items(): Retrieve all clothing items
- get_items_by_category(category): Filter by tops/bottoms/footwear
- get_items_by_color(color): Filter by color (Could-have)
- get_item_by_id(item_id): Fetch specific item details
- create_item(item_data): Add custom clothing item (Should-have: camera upload)

Database Interactions: Query operations on clothing collection

2.5. Data Layer Components

2.5.1. MongoDB Database

Role: Persistent storage for all application data

Collections:

users Collection:

```
{
  "_id": ObjectId("..."),
  "username": "string",
  "password_hash": "string",
  "created_at": ISODate("..."),
  "avatar_id": "string" (optional - Could-have)
}
```

outfits Collection:

```
{
  "_id": ObjectId("..."),
  "user_id": ObjectId("..."),
  "outfit_name": "string",
  "top_id": ObjectId("..."),
  "bottom_id": ObjectId("..."),
  "footwear_id": ObjectId("..."),
  "created_at": ISODate("..."),
  "updated_at": ISODate("...")
}
```

clothing Collection:

```
{
  "_id": ObjectId("..."),
  "category": "top|bottom|footwear",
  "color": "string",
  "image_path": "string",
  "description": "string" (optional),
  "is_default": boolean
}
```

Indexes:

- users.username: Unique index for fast login lookups
- outfits.user_id: Index for retrieving user's saved outfits
- clothing.category: Index for category filtering

3. Detailed Design

3.1. Architecture

The MisMatch application is constructed through layered integration where each tier builds upon the previous one.

The system components interact through well-defined interfaces:

Browser ↔ Web Server: HTTP requests/responses with session cookies
Controllers ↔ Business Logic: Function calls within Python application
Business Logic ↔ Data Access: MongoDB queries through pymongo driver
Web Server ↔ MongoDB: Connection pooling for efficient database operations

3.1.1. Foundation: Data Tier

MongoDB provides the persistence foundation:

- Stores three types of entities: users, outfits, clothing items
- Document-based storage allows flexible schema evolution
- Indexes ensure fast query performance
- Handles concurrent access from Flask application

3.1.2. Core Logic: Application Tier

Python web server orchestrates all business operations: It receives HTTP requests from the browser, routes them to the appropriate controllers, and coordinates interactions between the business logic services and data access layer.

Controllers handle the request-response cycle by validating user sessions, processing input data, and invoking the necessary services and DAOs to perform operations.

The application tier enforces business rules, manages application state through sessions, and ensures that data flows correctly between the presentation layer above and the data layer below.

Once processing is complete, the web server renders the appropriate HTML template with the requested data and sends the response back to the browser.

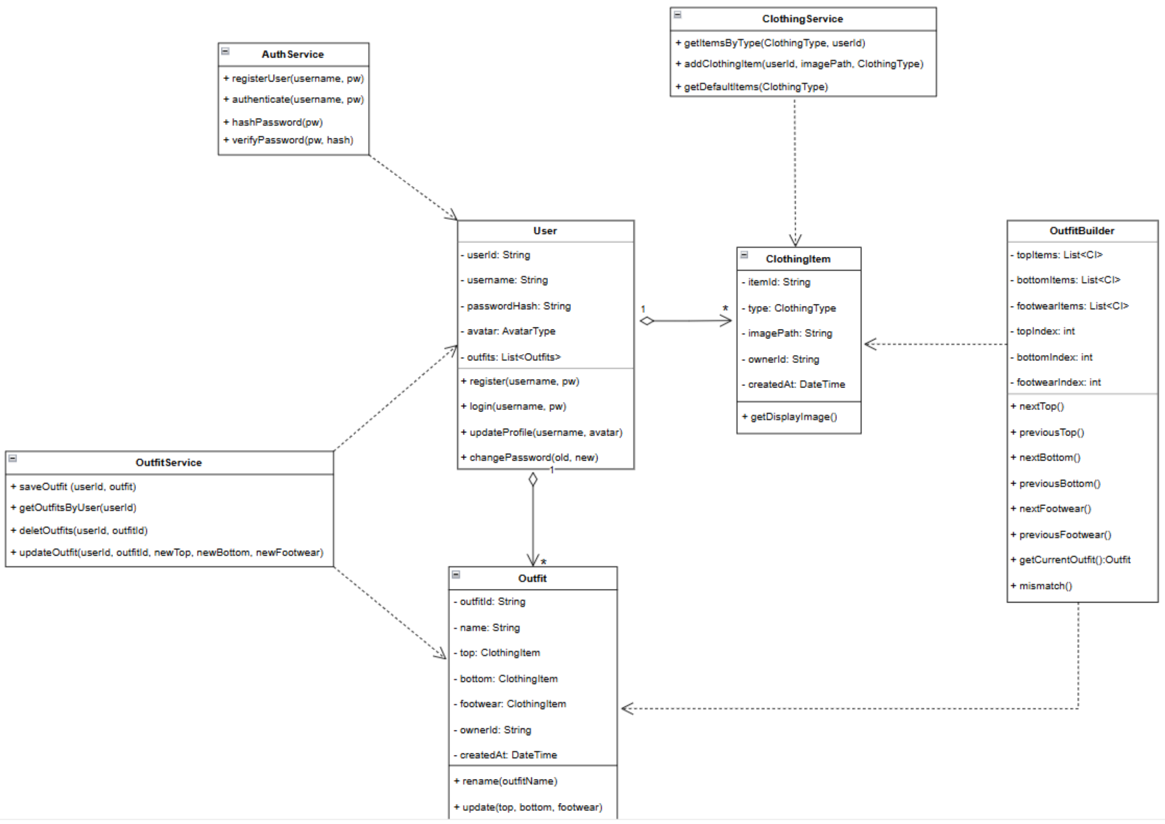
3.1.3. User Interface: Presentation Tier

Browser renders the interactive interface:

- HTML templates populated with data from web server
- CSS provides visual styling and layout
- JavaScript enables dynamic interactions without page reloads
- Clothing images displayed in interactive boxes with arrow controls

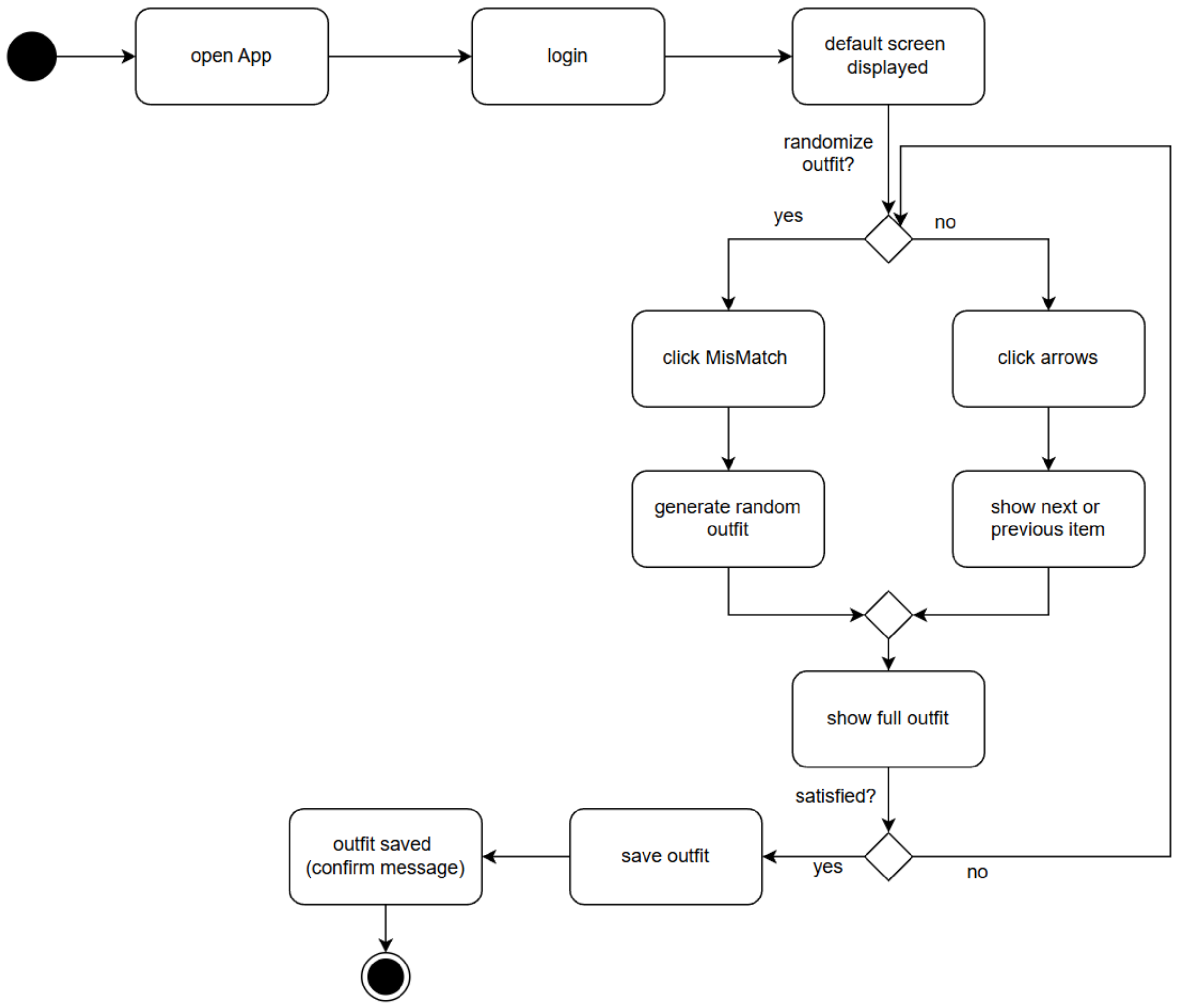
3.2. UML Diagrams

3.2.1. Class Diagram

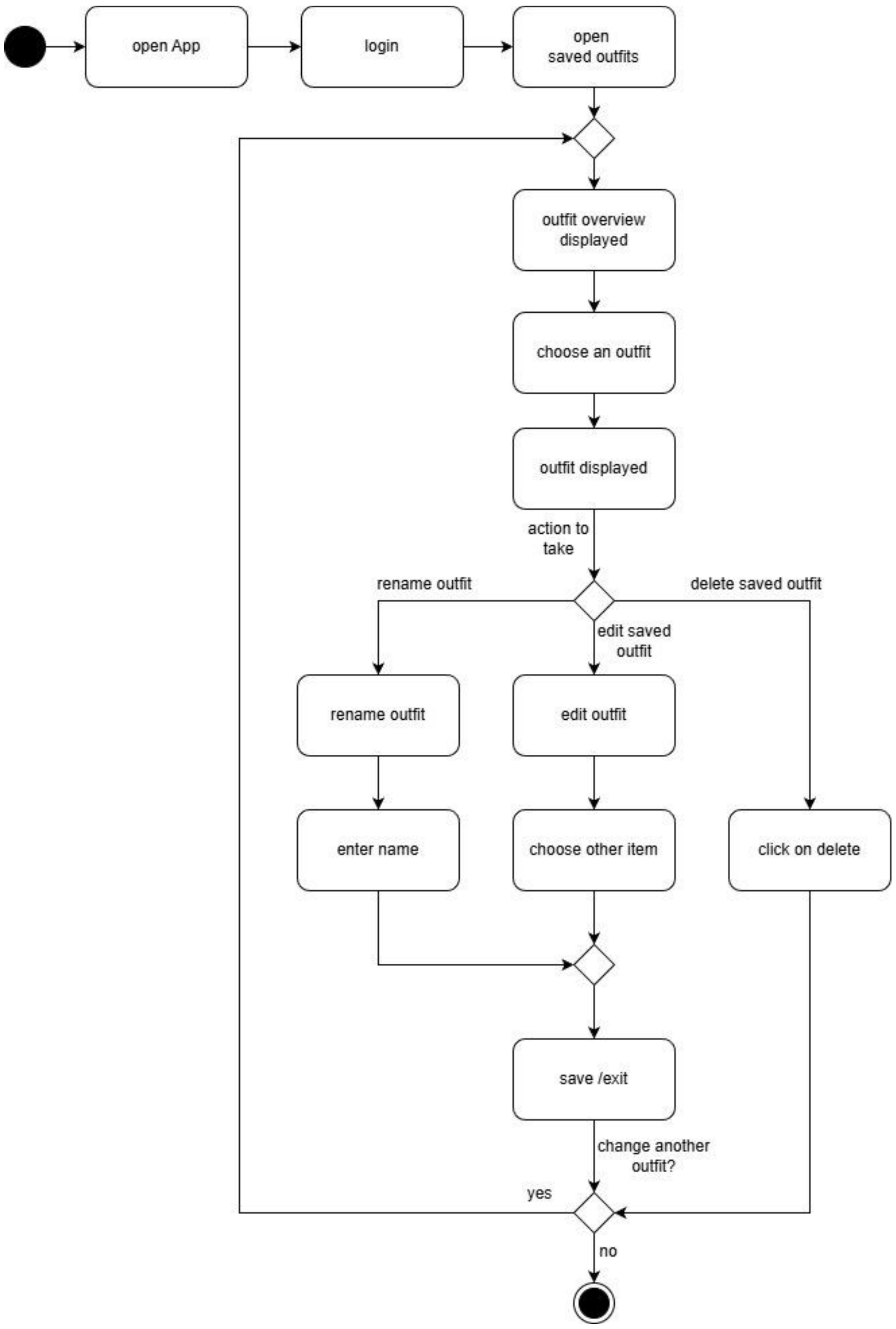


3.2.2. Activity Diagram

The diagram below shows a basic app workflow:

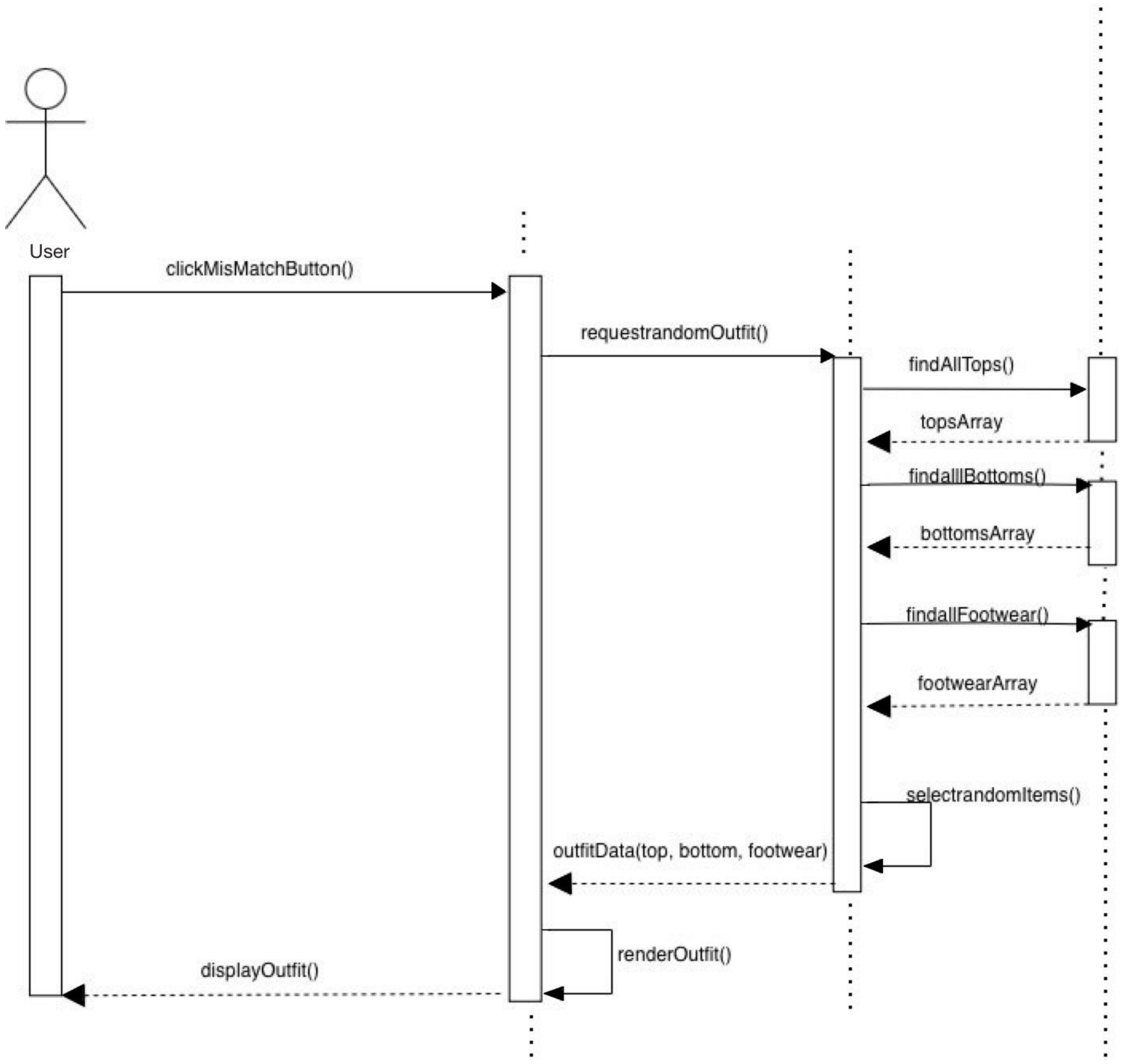


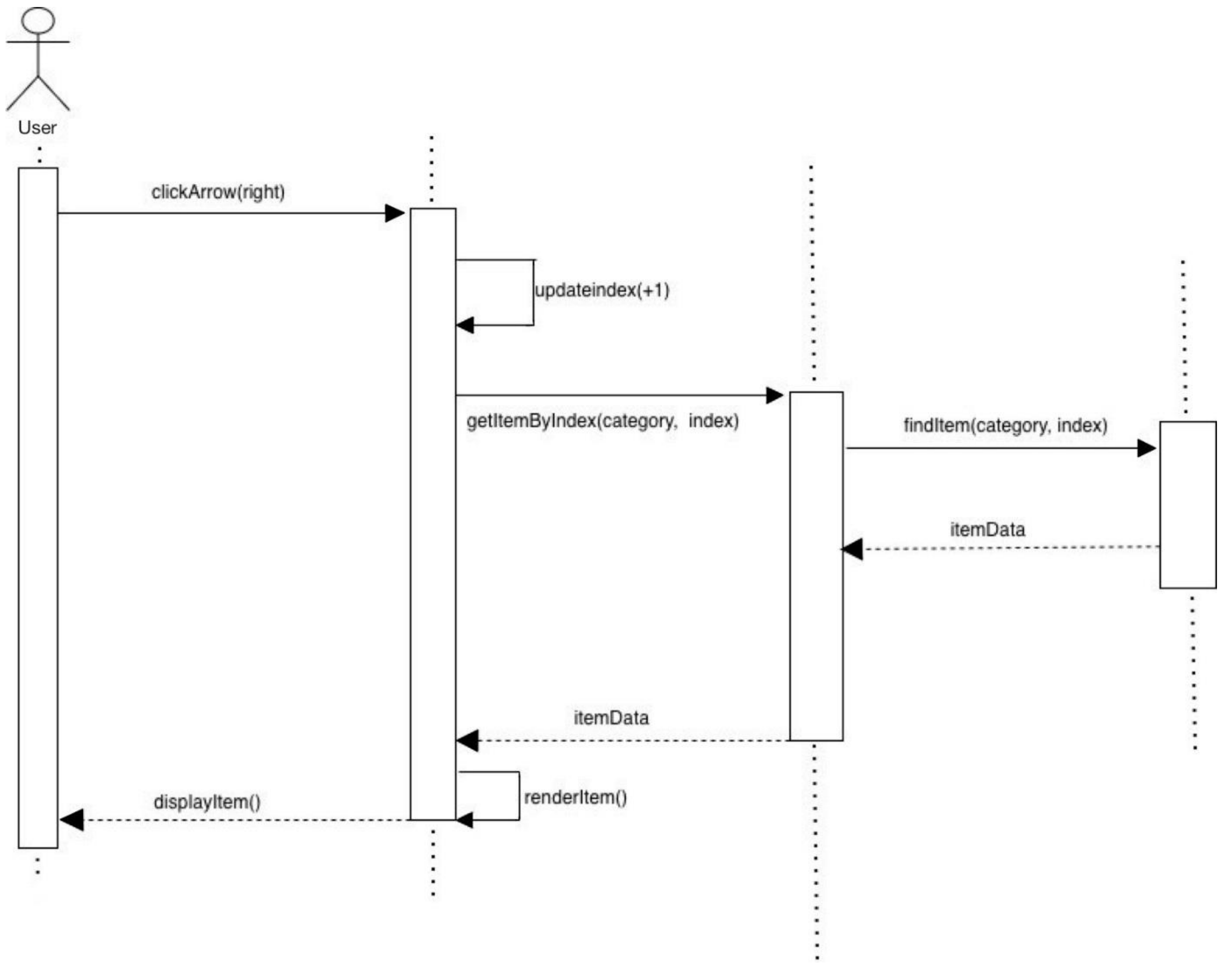
The diagram below shows the workflow of editing a saved outfit:



3.2.3. Sequence Diagram

This diagram shows how the system generates a random outfit by selecting one item from each clothing category when the user clicks the MisMatch button.





This diagram shows how the user navigates through clothing items using arrow buttons to view the next or previous item within a category.