Design Specification

FashionFinder

Team Members: Maria Alsamaien, Alex Gjeka, Angjelo Mana & Oloofa Kalid

Introduction

The FashionFinder Design Specification document serves as a comprehensive guide focusing on the architecture, design, and construction elements crucial for the development team's implementation of the FashionFinder website. It is created during the planning phase of the project to provide clear guidelines and technical insight into the development process. This document acts as a blueprint for the entire development lifecycle, from beginning to end, ensuring that the project goals and stakeholder expectations are met.

General Overview, Design Guidelines & Approach

General Overview

The FashionFinder website's design perspective centers around creating an engaging, user-friendly, and aesthetically pleasing platform for anyone interested in fashion. The core focus is to simplify the process of finding clothing for its users.

Assumptions

- Users of the FashionFinder website have an interest in fashion and are seeking a platform to discover and save fashion items.
- Users have access to a stable internet connection,
- Users can access the FashionFinder website from various devices, including desktops, laptops, tablets, and smartphones, ensuring a wide accessibility range.
- Users can interact with the website's features, such as creating and managing personal albums.

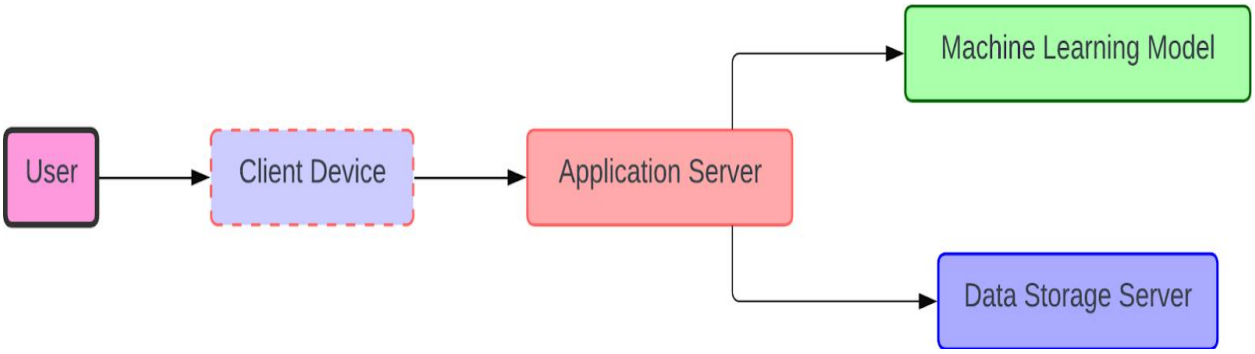
Constraints

- A weak, slow, or non-existent internet connection.
- The website server is not available or slow.
- The website must be designed to adapt to multiple screen sizes, ensuring a consistent and user-friendly experience across different devices, from smartphones to desktop computers.

Architecture Design

1. Hardware Architecture

Our AI FashionFinder application consists of several important aspects, including the user, the client device (such as a computer or mobile device), the application server, the machine learning model server, and the data storage server. Below is an overview of the hardware architecture:



The user interacts with the application through a client device, which could be a computer, tablet, or smartphone. Using the client device, the user accesses the AI Personal Designer application interface to input design preferences and interact with generated design suggestions.

For ideal performance when running our application, it is recommended to have hardware specifications meeting the following requirements: A minimum of 8GB of RAM ensuring smooth operation of application data. Also, a CPU with a speed of 2.5 GHz or higher provides enough power to execute tasks effectively. Lastly, having a GPU with at least 2GB of VRAM effective rendering graphics, enhancing the visual experience for users. Meeting these hardware specifications ensures that our application runs smoothly, providing users with a responsive experience.

Upon receiving user input, the client device sends requests to the application server. The application server hosts the core functionalities of the AI FashionFinder application, including the user interface, logic, and communication with other servers.

If the requested design suggestions or personalized recommendations are not readily available in the application server's cache or database, the application server communicates with the machine learning model server. This server hosts the trained machine learning models responsible for generating design suggestions based on user preferences and will return closely matching items.

At the same time, the application server may also interact with the data storage server to retrieve relevant user data, clothing options, and other resources necessary for generating design recommendations.

The machine learning model server generates the requested design suggestions; the application server then sends the results back to the client device after fetching any necessary data from the data storage server. Finally, the client device displays the design suggestions and recommendations to the user through the application interface, completing the interaction loop.

In summary, the AI FashionFinder application hardware architecture involves smooth communication between the user, client device, application server, machine learning model server, and data storage server to deliver personalized design recommendations and enhance user experience.

2. Software Architecture

Based on our project's requirements, we believe that a three-tier architecture pattern would be the most suitable option. Here are the reasons why:

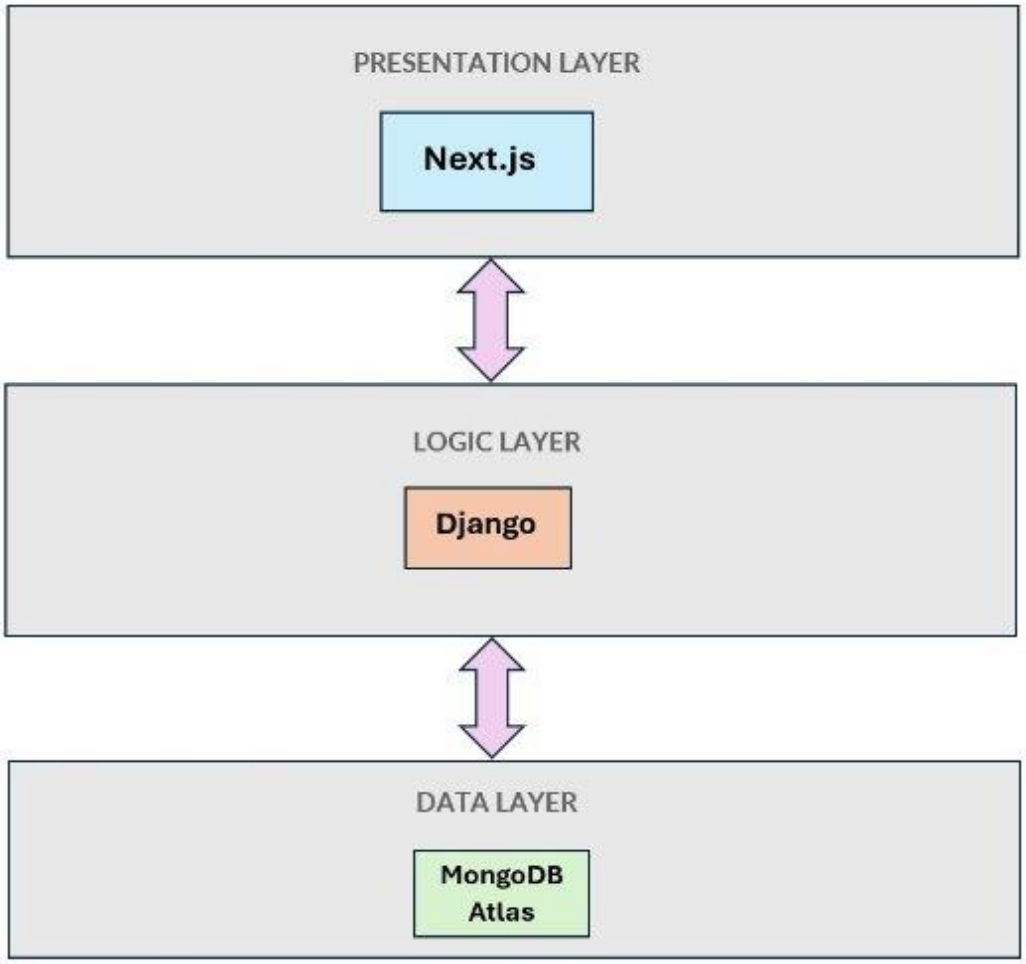
- Simplified Structure: a three-tier architecture streamlines the overall system structure by dividing it into three logical layers: presentation, application, and data. This clear separation of concerns can enhance the overall understandability of the system for developers, making it easier to maintain and extend.
- Scalability: a three-tier architecture allows for scalability at each tier. By independently scaling the presentation, application, and data layers, the system can efficiently handle increased loads without affecting other components.
- Performance Optimization: By consolidating certain functionalities into fewer components, a three-tier architecture can minimize latency and improve overall performance, especially in scenarios where microservices communication overhead might become a bottleneck.

The software system will be structured using a three-tier architecture, promoting simplicity, scalability, and maintainability. At its core, the system will comprise three distinct layers: the presentation layer, the business logic layer, and the data access layer. Each layer will serve a specific purpose, facilitating clear separation of concerns and enhancing overall system robustness.

The presentation layer will be implemented using Next.js and Tailwind CSS, providing users with a dynamic and responsive interface. This layer will focus solely on presenting information to users and gathering input from them.

The logic layer, powered by Django, will host the application's core functionality. It will handle business processes, orchestrate interactions between different parts of the system, and enforce business rules. This layer's design will prioritize scalability and flexibility, allowing it to adapt to changing business requirements seamlessly.

The data access layer will be responsible for interacting with the database, which will be MongoDB Atlas due to its scalability and compatibility with the project's dynamic data needs. This layer will ensure efficient retrieval and manipulation of data, optimizing performance and resource utilization.



Frontend Components

- User Interface (UI): Handles user interactions, displays forms for account creation, password change, signing in, album management, customer support, etc.
- Outfit Display UI: Displays outfit recommendations in image and link format.
- Album Management UI: Allows users to create, delete, and manage albums.
- Review System UI: Provides a user interface for leaving and displaying reviews.
- Customer Support: Allows the user to contact us with their inquiries.

- Chat Interface: Enables users to interact with the personal designer using natural language processing.
- Preferences UI: Facilitates the selection of brand preferences and budget input field.

Backend Components

- Authentication Module: Manages account creation, password change, and user sign-in.
- User Verification Module: Handles the verification process, sending verification emails, and checking the status.
- Database: Stores user accounts, preferences, albums, outfits, reviews, measurements, and messages.
- Album Management Logic: Manages the creation, deletion, and modification of user albums.
- Review System Logic: Manages the submission and retrieval of user reviews.
- Customer Support Module: Manages the submission and retrieval of customer support messages.
- Personal Designer Engine: The AI-driven component responsible for refining outfit searches and chatting with users.
- Outfit Matching Logic: Conducts the matching process for outfit recommendations.
- Preferences and Measurement Logic: Handles the storage and retrieval of user preferences and measurements.

3. Security Architecture

Before user passwords are saved to the database, they are first encrypted using bcryptjs. Also, upon user login, a json web token is generated and saved to their cookies, these will be checked for each HTTP request made to confirm that it is the user making these requests. This will be implemented within the middleware. These tokens are also set to HTTP only to ensure protection against cross site scripting.

4. Communication Architecture

Information between the client and server is done through HTTP requests and responses, we will be using axios to make requests and trigger server-side actions and fetching data from our backend APIs. To ensure secure communication between the client and server components, authentication tokens generated using the jasonwebtoken library will be leveraged as mentioned previously in the Security Architecture section. Our system will also be using the OpenAI API to extract key information from the user's prompt and structure

it into JSON format so that it may be used to query against the databases of other companies, once a match is found this will be returned to the UI.

5. Performance

Load Balancing

In our system, load balancing will be implemented to distribute incoming requests across multiple instances of servers to optimize resource utilization and ensure high availability. The Round Robin load balancing algorithm will be employed, ensuring that each server in the cluster gets an equal share of incoming requests. This approach promotes uniform distribution of the workload, preventing any single server from becoming a bottleneck. Load balancing will be achieved using a dedicated load balancer component within the system architecture, which will intelligently route traffic based on server availability and current load metrics.

Concurrency and Parallelism

To address concurrency and parallelism, the system will employ a multi-threaded architecture, allowing multiple tasks to be executed simultaneously. Key strategies include implementing thread safety through synchronization mechanisms to prevent race conditions, employing locks and mutexes where necessary. The use of parallel programming paradigms, such as parallel loops and asynchronous processing, will be considered to maximize the efficiency of parallelism. Additionally, careful design and testing will be conducted to avoid common pitfalls such as deadlocks and resource contention.

Response Time and Throughput

Quantifying response time and throughput metrics is crucial for ensuring optimal system performance. The document will define acceptable thresholds for response times under different load conditions. For instance, the system aims to achieve a response time of less than 500 milliseconds under normal operating conditions. Throughput benchmarks will be established to measure the system's capacity in terms of requests per minute (RPM). Load testing scenarios will be simulated to assess how the system performs under varying loads, ensuring that it meets or exceeds the predefined thresholds. Continuous monitoring tools will be employed to track response times and throughput metrics in real-time, allowing for proactive adjustments to maintain optimal system performance.

System Design

Use Cases

Use Case ID:	UC-1
Use Case Name:	User Registration and Profile Creation
Description:	This use case outlines the process of a new user registering for an account and creating a profile on the FashionFinder website.
Actor:	User
Trigger:	A new user accesses the FashionFinder website and clicks the button “Create Account”
Preconditions:	• The FashionFinder website is accessible and operational. • The user has internet connectivity and access to a web browser. • The user has not registered for an account previously.
Postconditions:	• The user successfully creates a FashionFinder account, and a profile is established in the system. • The user gains access to personalized features and functionalities offered by FashionFinder.
Normal Flow:	1. User enters the FashionFinder website. 2. User clicks on the "Create account" button. 3. System displays the registration form. 4. User fills in email, password, and confirms password. 5. User submits the form.

	6. System validates user input. 7. System checks email uniqueness. 8. If all validations pass, the system creates the account. 9. User receives an email verification link. 10. User is redirected to the login page.
Alternative Flow:	1. User enters the FashionFinder website. 2. User tries to login but doesn't have an account. 3. User presses “create account” link under password input box. 4. System displays the registration form. 5. User fills in email, password, and confirms password. 6. User submits the form. 7. System validates user input. 8. System checks email uniqueness. 9. If all validations pass, the system creates the account. 10. User receives a confirmation message. 11. User is redirected to the login page to successfully login.

Exceptions:	• If the user enters incomplete or invalid information:
	  ○ System prompts the user to correct the errors and resubmit the form.
	• If the email address provided during registration is already associated with an existing account:
	  ○ System notifies the user to choose a different email address or attempt to recover the existing account.
	• If the user attempts to register with a weak password:
	  ○ System prompts the user to choose a stronger password that meets the security requirements.

Assumptions	• Users have average internet browsing skills and can navigate web forms. • Users provide accurate and verifiable information during the registration process. • The FashionFinder website has adequate security measures in place to protect user data and privacy.

Use Case ID:	UC-2
Use Case Name:	Adding outfits to albums
Description:	This use case outlines the process by which a registered user adds outfits to an album they created.
Actor:	User
Trigger:	Logged in user clicks the heart icon on the desired outfit.
Preconditions:	• The FashionFinder website is accessible and operational. • The user is logged into their registered account.

	• The user has found an outfit they'd like to add through communicating with personal designer • The user already created the album they'd like to save to.
Postconditions:	• The outfit is successfully added to the user's desired album. • The user can view, share, and delete items within their album.
Normal Flow:	1. User navigates to the FashionFinder website and logs into their registered account. 2. User will press the heart icon that appears when the outfit is returned 3. A modal will appear with the names of the user's albums, user will click the name of the album to save to 4. User will click the confirm button to confirm saving to that album
Exceptions:	• If the outfit failed to save, a message will appear asking the user to try again
Assumptions:	• Users are familiar with basic website navigation and interaction principles. • The FashionFinder website provides clear visuals and user-friendly interfaces for adding items to the album page. • The website effectively manages user sessions and data storage to ensure the integrity of album page information.

Use Case ID:	UC-3
Use Case Name:	Searching for Clothes

Description:	This use case illustrates the process by which a registered user searches for clothes on the FashionFinder website.
Actor:	User
Trigger:	A registered user writes their clothing preferences in the chat box and clicks enter.
Preconditions:	• The FashionFinder website is accessible and operational. • The user is logged into their registered account, although this feature can also be available to non-registered users. • The user has access to the search/chat functionality or navigates to the main page of the website.
Postconditions:	• The user successfully finds relevant clothing based on their search query or browsing preferences. • The user may choose to save or explore additional details about the discovered clothing.
Normal Flow:	1. User accesses the FashionFinder website and navigates to the search bar or the fashion trends section. 2. User inputs specific keywords, phrases, or topics related to the desired fashion trends or styles. 3. System processes the user's search query and retrieves relevant fashion trends, styles, and inspirations matching the search criteria. 4. User reviews the search results presented by the system 5. User clicks on a specific trend or style to view additional details, such as outfit ideas, price information, or clothing brand.

	6. User explores multiple trends and styles, refining their search criteria as needed. 7. User may choose to save favorite trends or styles for future reference or exploration to the albums page.

Exceptions:	• If the user's initial search query does not yield satisfactory results:
	  ○ User refines their search criteria by adjusting keywords or filters.
	• If the user encounters irrelevant or unrelated search results:
	  ○ User modifies their search query to better align with their intended fashion interests.
	• If the user experiences technical difficulties or system errors during the search process:
	  ○ User may retry the search or contact customer support for assistance.

Assumptions:	• Users understand fashion terminology and trends. • The FashionFinder website has efficient search algorithms to deliver accurate and relevant search results. • The search functionality is accessible and user-friendly across various devices and screen sizes.

Use Case ID:	UC-4

Use Case Name:	Creation of User Albums
Description:	This use case covers the process by which a registered user creates an album.
Actor:	User
Trigger:	A registered user goes to the album page and clicks “create album” button.
Preconditions:	• The user is logged into their account.
Postconditions:	• The user successfully creates an album. • Users can open the album by clicking it.
Normal Flow:	1. User clicks album link in the navigation bar. 2. User clicks “create album” button. 3. User provides a unique name for the album. 4. User confirms the creation through pressing the confirm button in the modal
Exceptions:	• If the user does not enter a name for the album, the “Create” button will be disabled • If the user tries to create a duplicate album name ○ User will get an error message stating, “Album name already exits”
Assumptions:	• Users understand the purpose and use of organizing clothing into albums for reference.

Use Case ID:	UC-5

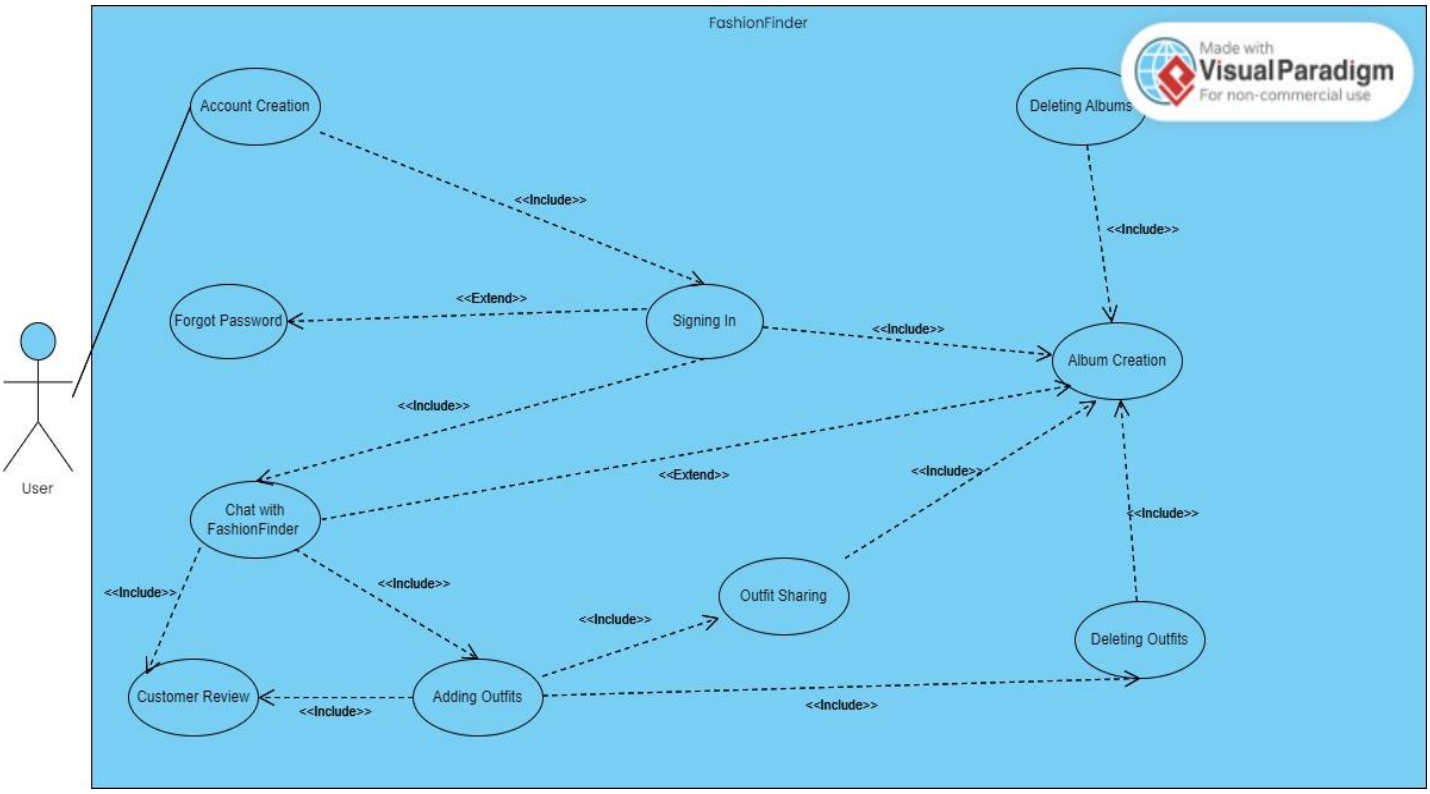
Use Case Name:	Deletion of User Albums
Description:	This use case covers the process by which a registered user deletes an album.
Actor:	User
Trigger:	A registered user goes to the album page and clicks the delete icon.
Preconditions:	• The FashionFinder website is accessible and operational. • The user is logged into their registered account. • The user has at least one album created.
Postconditions:	• The user successfully deletes an album. • The user's albums are effectively organized and reflect their desired collections of fashion items.
Normal Flow:	1. User accesses the FashionFinder website and logs into their registered account. 2. User navigates to the album section within their account settings or dashboard. 3. User initiates the process of deleting a new album. 4. User confirms the deletion through the designated action buttons or prompts. 5. System updates the user's album collection to reflect the addition of the album.
Exceptions:	• If the user decides to cancel the album deletion process midway: ○ User is provided with options to confirm or discard the

	changes before finalizing the action.
Assumptions:	• Users understand the purpose and use of organizing outfits into albums for easier access and management. • The album management feature is intuitive and user-friendly, offering clear instructions and prompts for deleting an album.

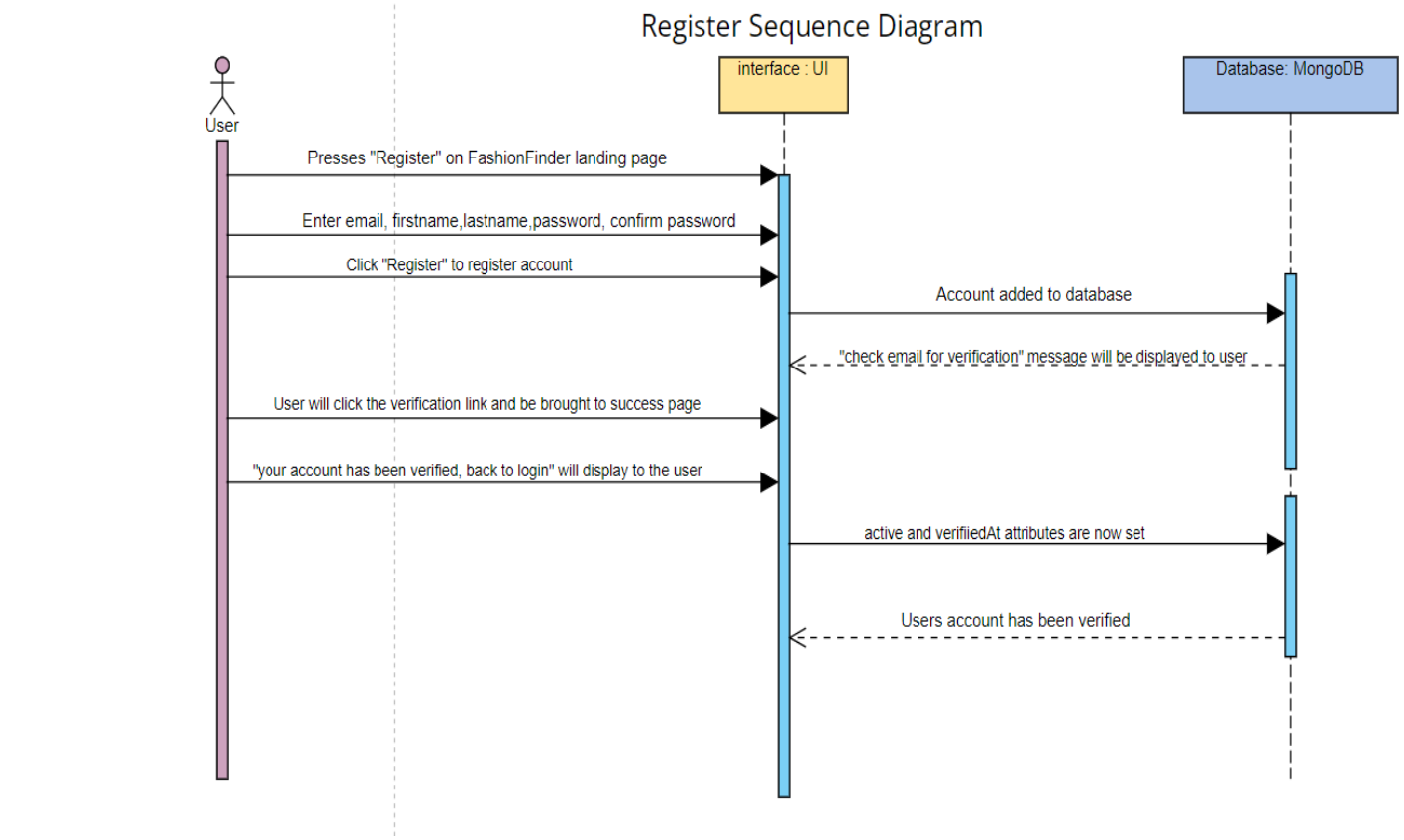
Use Case ID:	UC-6
Use Case Name:	Customer Support Services via Contact Us Page
Description:	This use case outlines the process by which website users utilize the "Contact Us" page to access customer support services provided by the FashionFinder website.
Actor:	User
Trigger:	The website user clicks “contact us” at the top of the main page.
Preconditions:	• The FashionFinder website is accessible and operational. • The "Contact Us" page is prominently displayed and accessible from the website's main navigation menu.
Postconditions:	• The user successfully submits a message to the FashionFinder customer support team. • The customer support team receives and processes the message and provides a response.

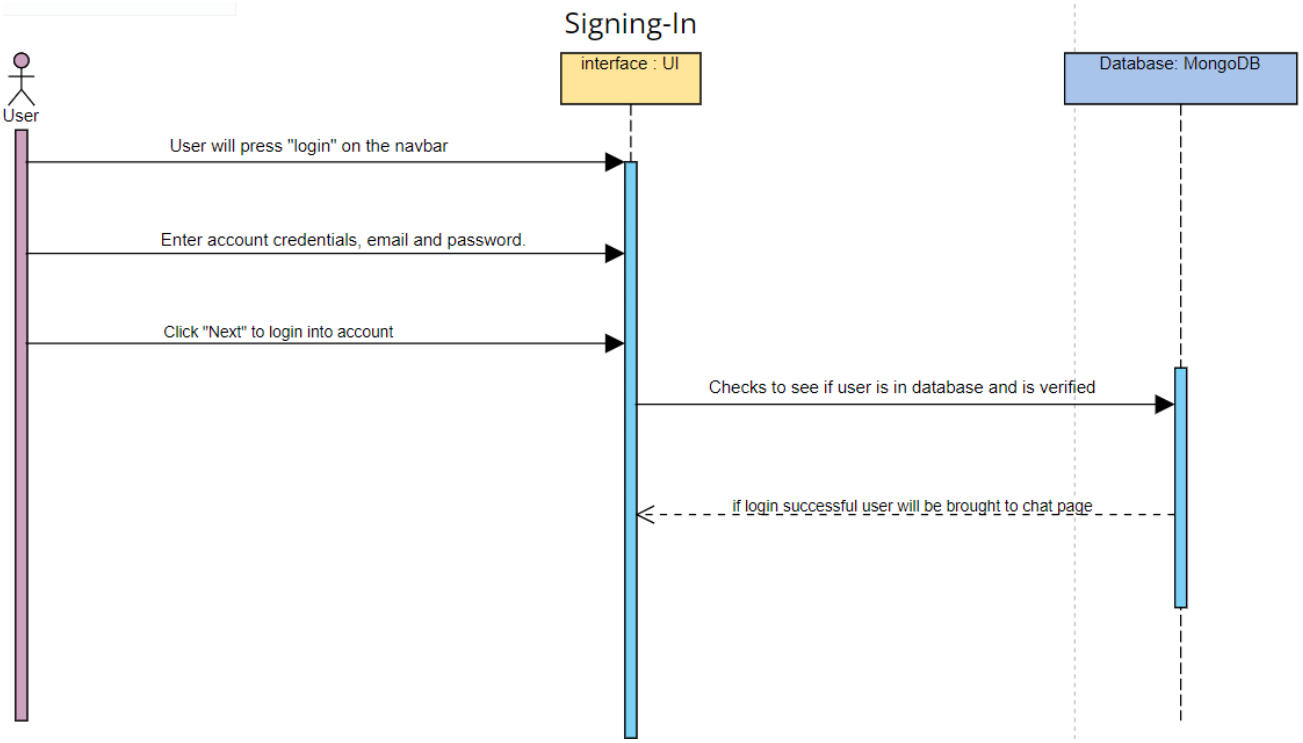
Normal Flow:	1. User navigates to the FashionFinder website and locates the "Contact Us" page. 2. User clicks on the "Contact Us" button to access the contact form or contact details. 3. User selects the appropriate contact method, such as filling out a contact form, then sending a message. 4. System acknowledges the submission of the message. 5. FashionFinder customer support team receives the submitted message. 6. Customer support team acknowledges and views the message and replies to the user if needed. 7. User receives a response from the FashionFinder customer support team.
Exceptions:	• If the user encounters technical issues or errors while submitting the contact form: ○ user refreshes the page or tries submitting the form again.
Assumptions:	• The "Contact Us" page provides clear instructions for website visitors to reach out to the customer support team. • FashionFinder's customer support team is ready to respond to various inquiries, from website users. • Customer support response times may vary based on the nature and complexity of the message.

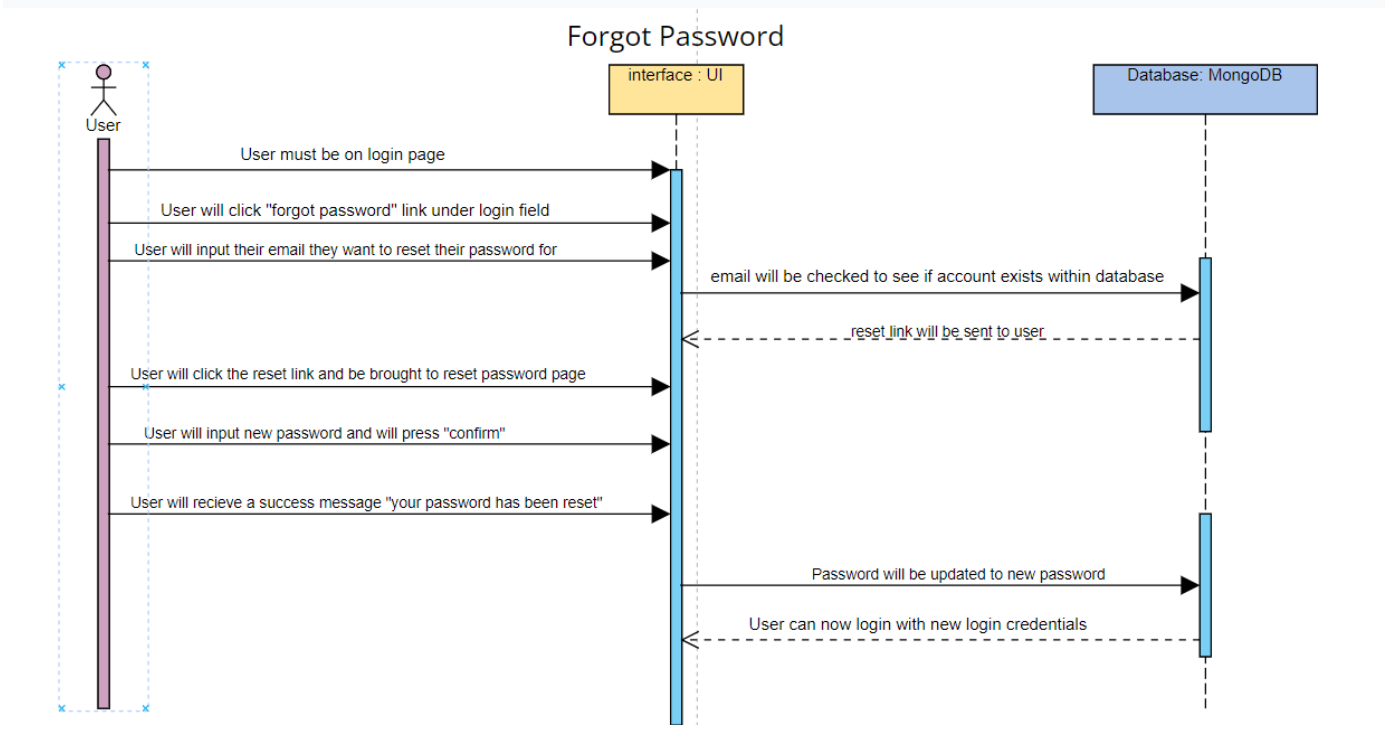
Use Case Diagram

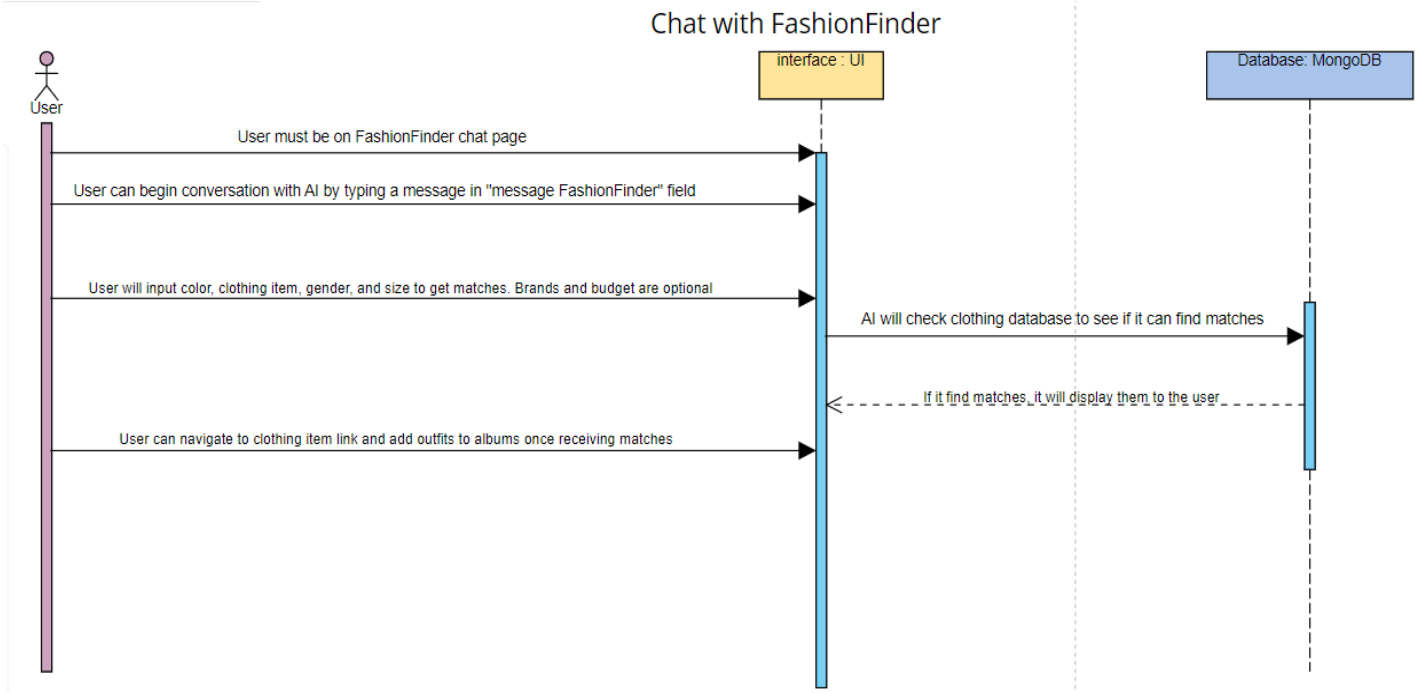


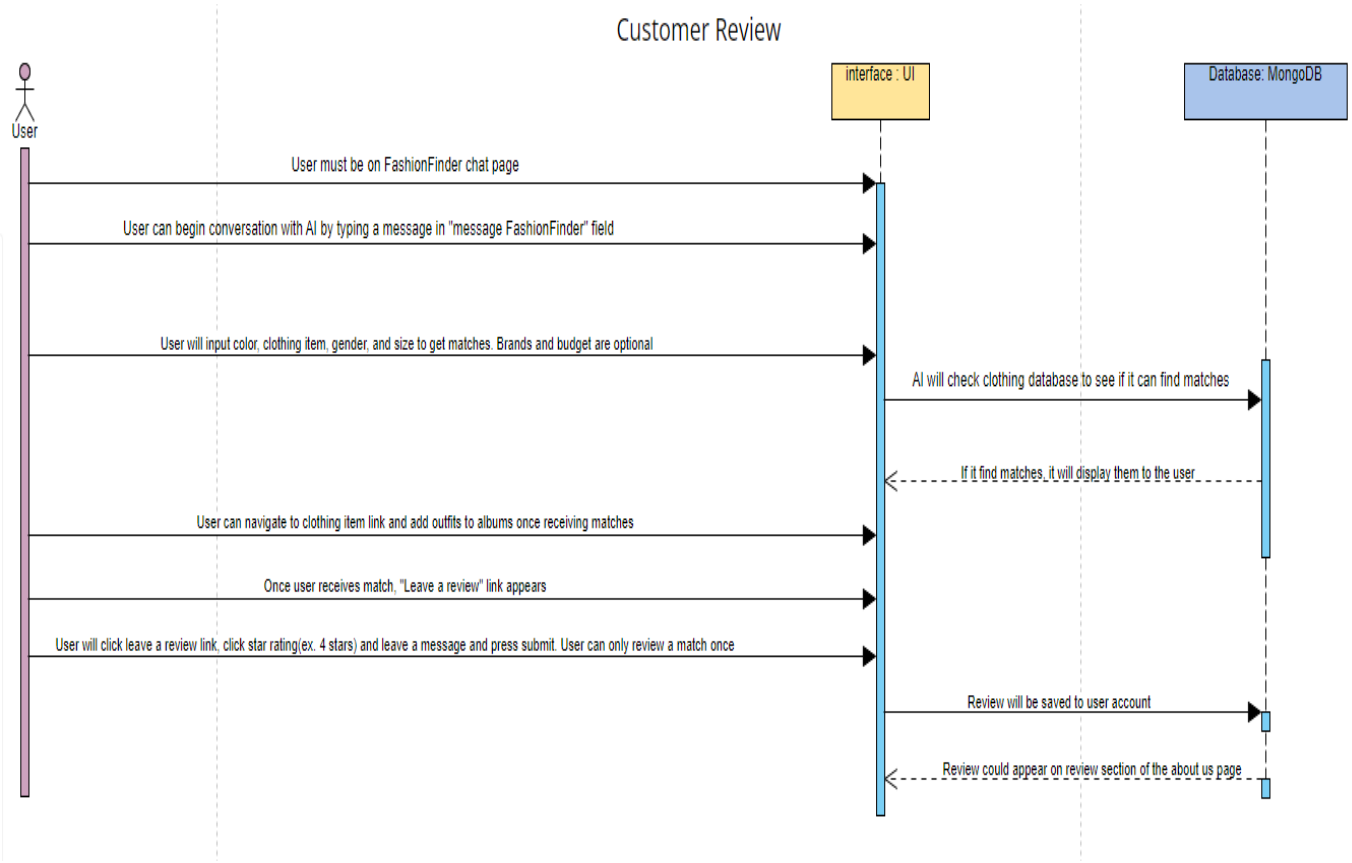
Sequence Diagram

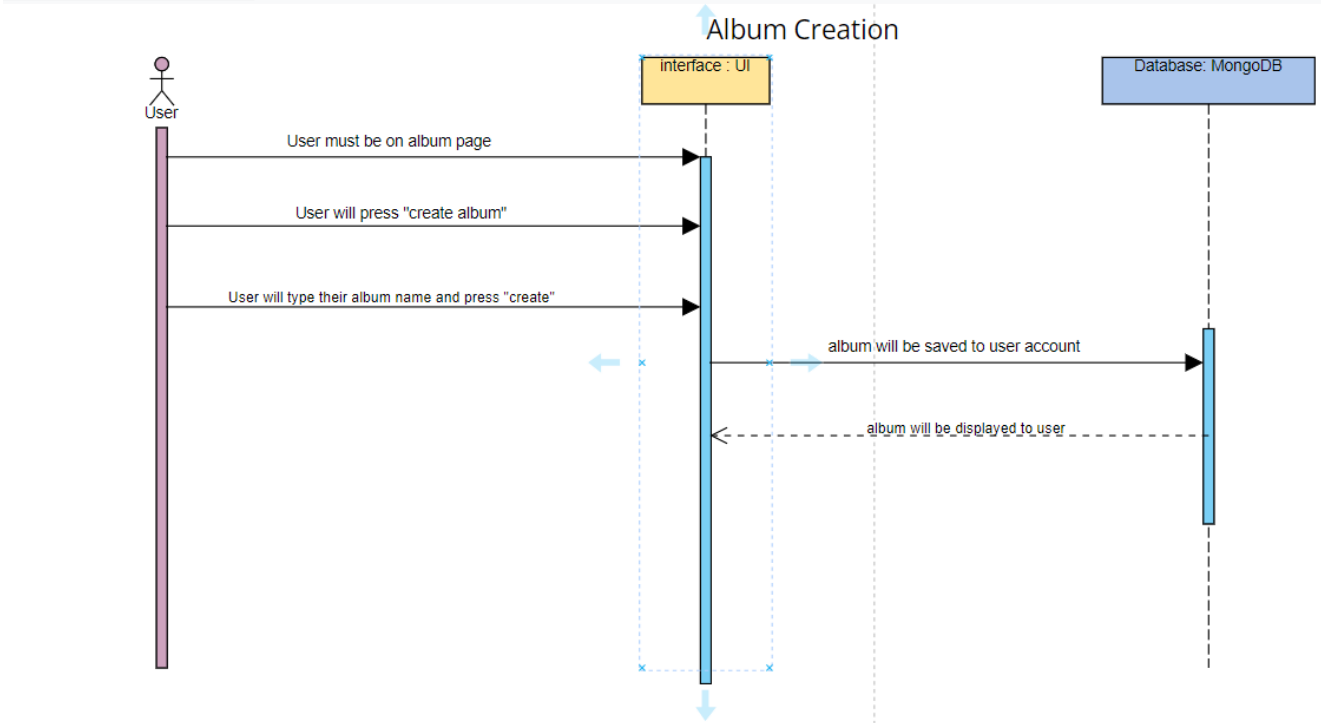


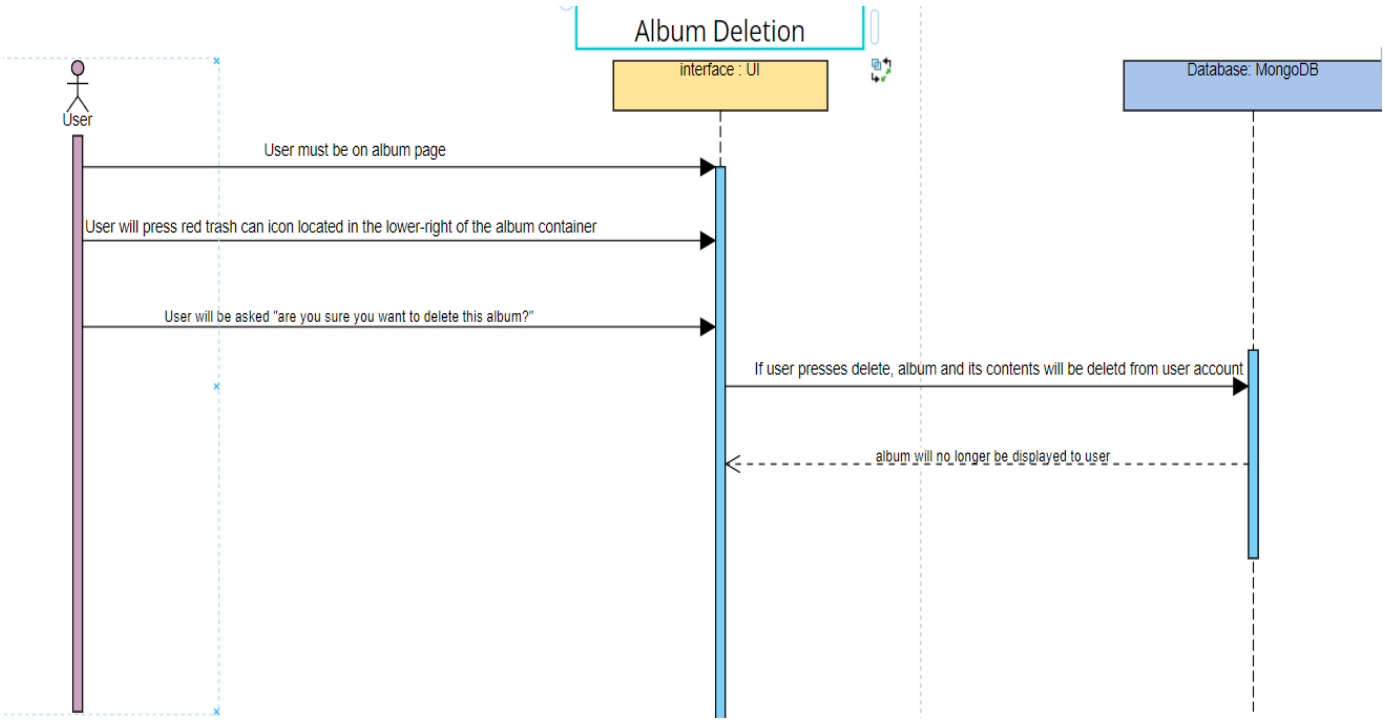


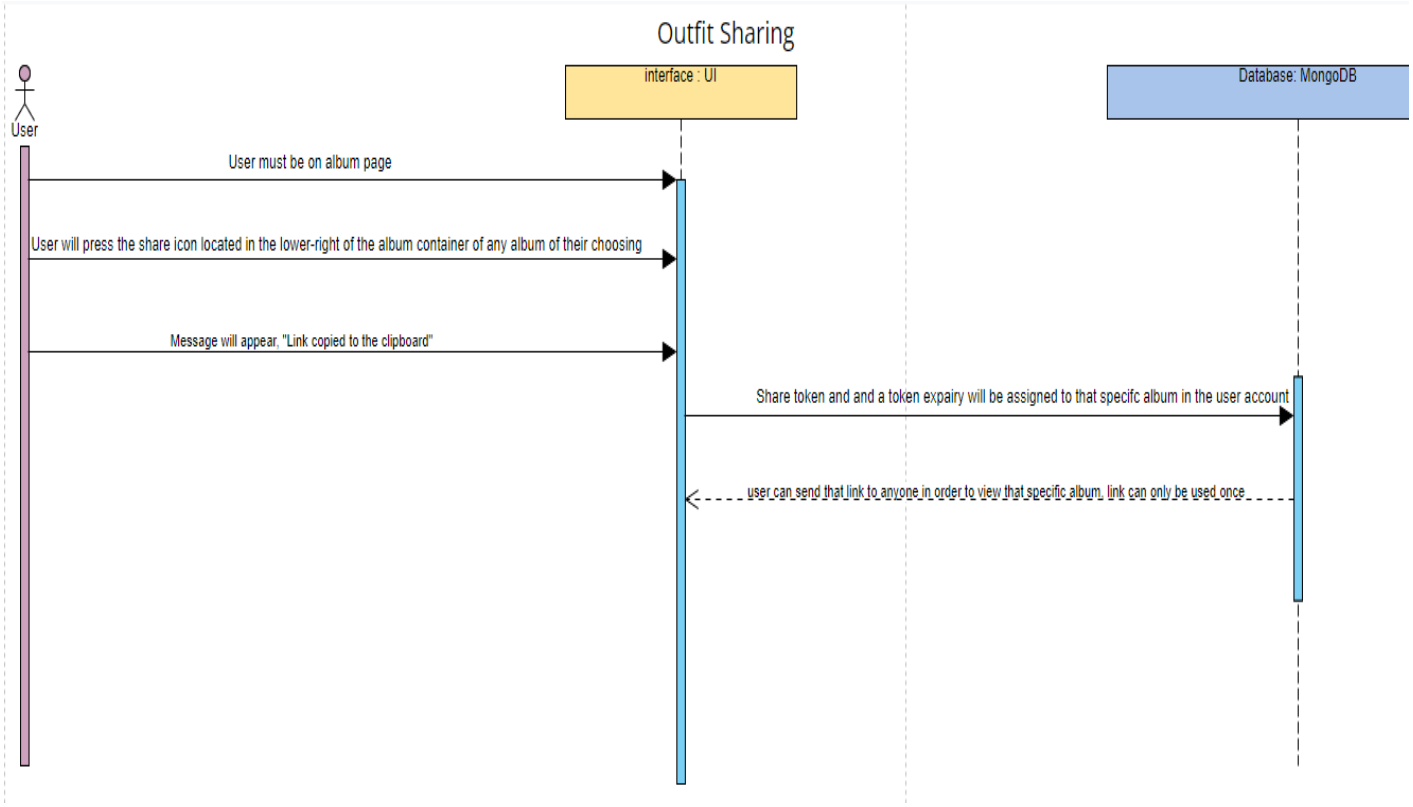

Forgot Password
User
interface : UI
Database: MongoDB
User must be on login page
User will click "forgot password" link under login field
User will input their email they want to reset their password for
email will be checked to see if account exists within database
reset link will be sent to user
User will click the reset link and be brought to reset password page
User will input new password and will press "confirm"
User will recieve a success message "your password has been reset"
Password will be updated to new password
User can now login with new login credentials


Chat with FashionFinder
User
interface : UI
Database: MongoDB
User must be on FashionFinder chat page
User can begin conversation with AI by typing a message in "message FashionFinder" field
User will input color, clothing item, gender, and size to get matches. Brands and budget are optional
AI will check clothing database to see if it can find matches
If it find matches, it will display them to the user
User can navigate to clothing item link and add outfits to albums once receiving matches


Customer Review
User
interface : UI
Database: MongoDB
User must be on FashionFinder chat page
User can begin conversation with AI by typing a message in "message FashionFinder" field
User will input color, clothing item, gender, and size to get matches. Brands and budget are optional
AI will check clothing database to see if it can find matches
If it find matches, it will display them to the user
User can navigate to clothing item link and add outfits to albums once receiving matches
Once user receives match, "Leave a review" link appears
User will click leave a review link, click star rating(ex. 4 stars) and leave a message and press submit. User can only review a match once
Review will be saved to user account
Review could appear on review section of the about us page

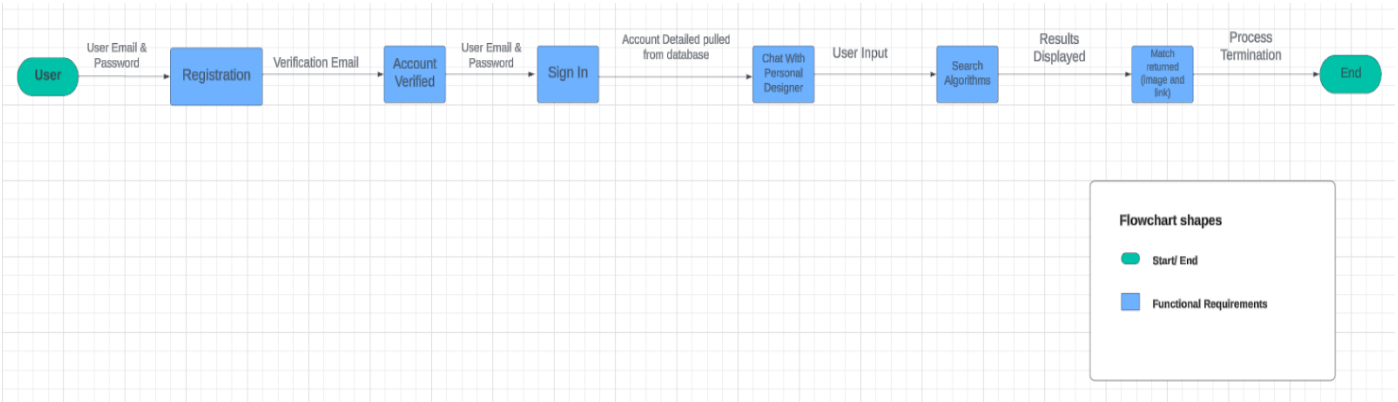

Album Creation
User
interface : UI
Database: MongoDB
User must be on album page
User will press "create album"
User will type their album name and press "create"
album will be saved to user account
album will be displayed to user


Album Deletion
User
interface : UI
Database: MongoDB
User must be on album page
User will press red trash can icon located in the lower-right of the album container
User will be asked "are you sure you want to delete this album?"
If user presses delete, album and its contents will be deletd from user account
album will no longer be displayed to user

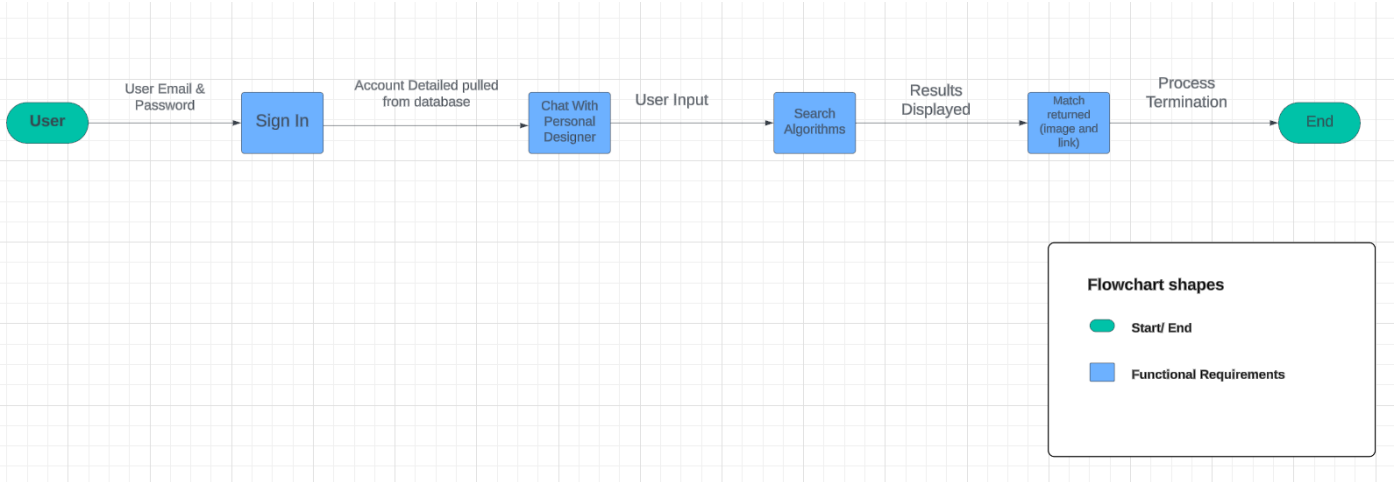


Data Flow Diagram

Here is a data flow diagram shown through a user that starts with registering.

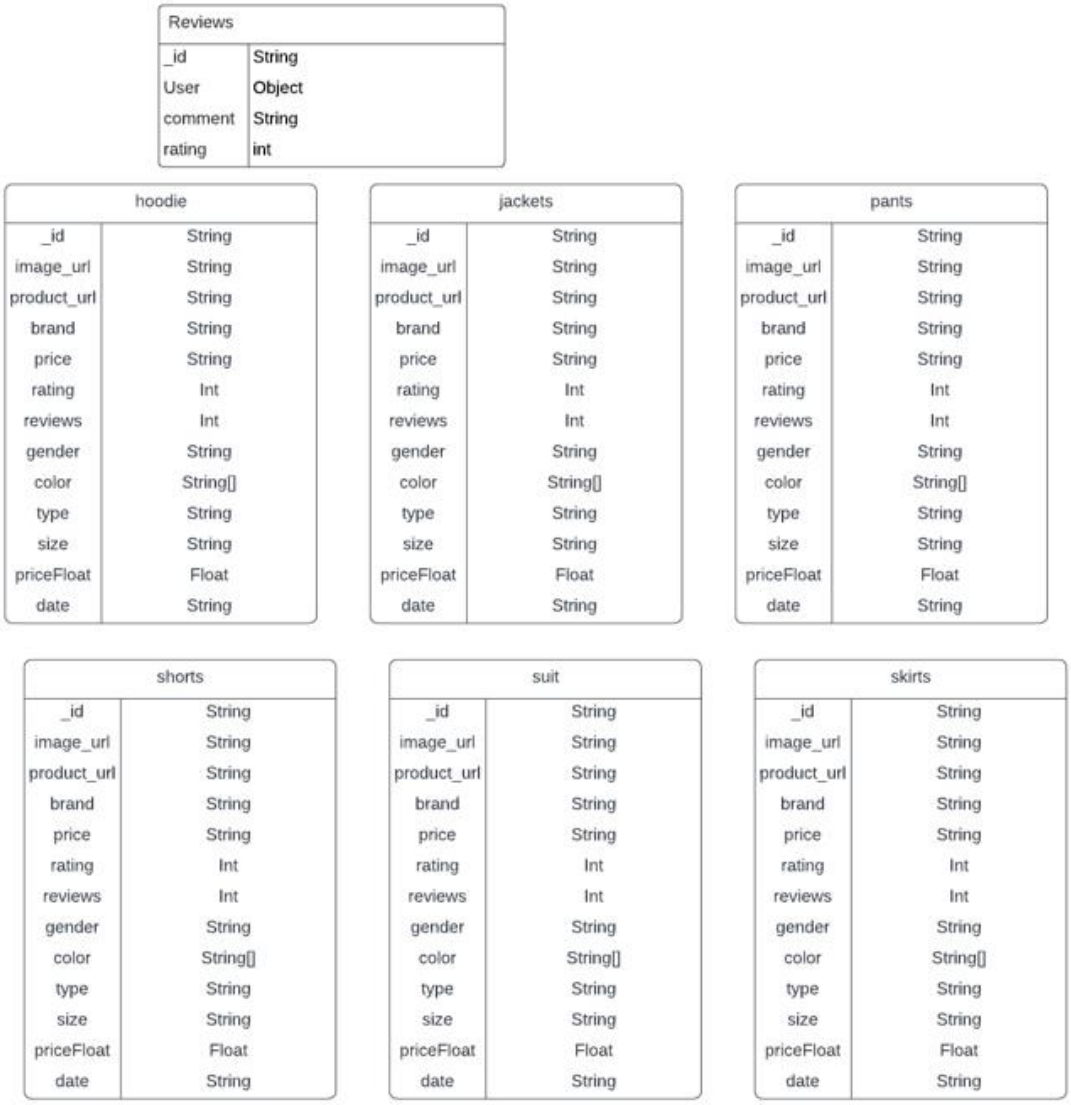


Here is a data flow diagram shown through a user that's already registered and is going through signing in.



Database Design


User
_id String
email String
password String
firstname String
lastname String
brands String[]
resetPasswordToken String
resetPasswordTokenExpiry Date
verifyToken String
verifyTokenExpiry Date
activatedAt Date
active Boolean
albums String[]
albumName string
dateCreated Date
timesOpened Int
shareToken String
shareExpiry Date
outfits String[]
outfitUrl String
imageUrl String
price String
color String
brand String
rating String
dateAdded Int
dress
_id String
image_url String
product_url String
brand String
price String
rating Int
reviews Int
gender String
color String[]
type String
size String
priceFloat Float
date String
hoodie
_id String
image_url String
product_url String
brand String
price String
rating Int
reviews Int
gender String
color String[]
type String
size String
priceFloat Float
date String
top
_id String
image_url String
product_url String
brand String
price String
rating Int
reviews Int
gender String
color String[]
type String
size String
priceFloat Float
date String
shorts
_id String
image_url String
product_url String
brand String
price String
rating Int
reviews Int
gender String
color String[]
type String
size String
priceFloat Float
date String



Application Program Interfaces (APIs)

FashionFinder uses numerous APIs to accommodate various functionalities. The register API first checks to see if the user already exists within the database. If this user does not exist, it will then ensure that all required fields have been filled, the password meets requirements, the confirm password matches the password, and if the email is in a valid format. If all of this is met, it will then hash the password, which is then followed by saving this user and information to the database. Upon any error within these fields, the user will be met with an error letting them know which of the following requirements have not been met and why they are seeing an error.

The login API will also have checks similar to that of the register API, being it will first check if the user exists, and then if the email is in a valid format. If the user exists and the email is

in valid format, the password is then matched against that of the user with the email they entered, if they match, a token is then created using jasonwebtoken and saved to the user's cookies. This token contains the user id and email. This token also has an expiration of 1 day. Any error here will also result in an error message to the user letting them know what the issue is.

The logout API simply removes the token from the user cookies. It first clears the value from the token and changes the expiration date to a date in the past, essentially deleting the cookie upon clicking the logout button immediately.

The email API is used for facilitating email verification for users. When a POST request is received with the user's email, this endpoint will connect with the database and checks if there is an existing user with that email. If there is an existing user, then a verification token is created and stored along with its expiry. A verification email will then be sent to the user, if sending the email fails then the token and its expiry time are removed from the user in the database.

The verify-email API handles confirming that is the user attempting to create an account with FashionFinder. When the verification token is received, the endpoint will connect to the database and check if there is a matching hashed token within the database and ensure it is not expired, if these conditions are met, then the person's account is activated. If there is not an existing token, or the token is expired, a response will be sent back saying that it is an invalid token, or the token has expired.

The reset API is responsible for updating a user's password upon changing it within the database. The API will extract the email and new password from the payload. It will then connect to the database and retrieve the record of that user within the database and update the change the password to a hashed value of the new password.

The forgot API handles the functionality for the forgot password feature. It follows a similar implementation as the reset API but also sends a confirmation email to the user to ensure it is them attempting to change their password. Again, this API will extract the email from the payload and checks the database to see if this is an existing email within the FashionFinder database. If the user is found, a hashed reset token is generated and is given an expiration time of 6 minutes. These are then saved to the users record within the database. A reset URL is creating using the token and sends a password reset email using that user's email address. Once the user clicks this link, the password reset process is initiated.

User Interface Design

Every page of the FashionFinder website will have the navigation bar at the top, to do this we must create a Header.js component and add it to the layout file within our next.js project. The

navigation bar will have links to different pages depending on the login status of the user. The links shown to all users regardless of login status are: FashionFinder (home), Contact Us, About, and Albums. Logged out users will also see Sign in and Create Account, whereas logged in users will see Profile and Logout.

Each page will have a background color of a slightly light gray, hash value of #f3f4f6. Each page is designed to be as simple as possible to not confuse the user. We will try to minimize unnecessary text on all pages so that the user does not feel bombarded with information. Each page must only have buttons or information that directly contributes to that page's functionality.

A similar color scheme is also presented across all pages, being a mix of white, black, gray, red for error messages, and for album creation the confirmation buttons for deleting and creating will be in blue. All text, buttons, and elements on the pages must have enough contrast so that they do not blend in. Bright colors should be avoided as they may lead to straining the user’s eyes, distracting the user from the purpose of the page/functionality, and clashes with content.

In the case where users create an album, want to delete an album, or logout, a modal will appear where the user can confirm or cancel their request. When this occurs, a light gray overlay will cover the page and be set at a higher z axis so that the rest of the page can be accessible to the user. The modal itself will be set to the same z axis so that it will be on level with the gray overlay, making it still accessible to the user. Once the modal is closed, the page will return to normal.