VIRTUAL WARDROBE MANAGEMENT SYSTEM

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By
RIA SINGLA 191232

Under the supervision of
Dr. DEEPAK GUPTA
to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan 173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Virtual Wardrobe Management System"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 2023 to June 2023 the supervision of **Dr. Deepak Gupta** of Computer Science And Technology.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Ria Singla (191232)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor Name: Prof. Dr. Deepak Gupta
Designation: Assistant Professor (SG)
Department name: Computer Science & Engineering
Dated:

PLAGIARISM CERTIFICATE

ACKNOWLEDGEMENT

Firstly, we would like to express our heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

We are really grateful and wish our profound indebtedness to Dr. Vivek Kumar Sehgal, Professor & HOD of the Department of CSE & IT, Jaypee University of Information Technology,Waknaghat.

Deep Knowledge & keen interest of my supervisor has helped me a lot to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non- instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Regards,
Ria Singla
191232

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

S.No.	Abbreviation	Full Form
1.	API	Application Programming Interface
2.	CRUD	Create, Read, Update, Delete
3.	DTO	Data Transsfer Object
4.	ORM	Object-Relation Mapping
5.	SQL	Structured Query Language
6.	MVC	Model-View-Controlller
7.	ASP.NET	Active Server Pages .NET
8.	EF	Entity Framework
9.	UI	User Interface
10.	UX	User Experience
11.	CSS	Casscading Style Sheets
12.	HTML	Hypertext Markup Language
13.	REST	Representational State Transfer
14.	JSON	JavaScript Object Nottation
15.	HTTP	Hypertext Transferr Protocol
16.	IDE	Integrated Development Environment
17.	API	Aplication Programming Interface
18.	HTTPS	Hypertext Transfer Protocol Seccure
19.	JWT	JSON Web Token
20.	OOP	Object-Oriented Programmiing

`

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The Virtual Wardrobe Clothing Management System, an online platform, allows users to maintain and produce outfits remotely. It features different microservices and provides variety of services, like outfit creation, search and filtering, user authentication and authorizaion, and clothing item management. Users may add theirr clothing pieces, search and filter them on various factors such as color, size, and kind, and then combine thhese things to create outfits.

The system employs a recommendation engine to generate outfit suggestions based on the user's own style, the occasion, and the weather. To present tailored outfit possibilities, the suggestion engine considers the user's previous clothes decisions, clothing preferences, and styles. Users may also store and share their favourite costumes.

Another feature of the Virtual Wardrobe Management System is an outfit creator, which allows users to create their outfits up to well ahead of time of occassion, as well as a packing list generator, which recommends appropriate items for an ocassion depending on the atmosphere and location.

The project was built using technologies such as ASP.NET Core, ReactJS, and Microsoft Azure cloud service. Because it’s incredibly extensible, safe, and simple to use, it’s the ideal choice for anybody who wants to manage their wardrobe and design outtfits online.

CHAPTER-1 INTRODUCTION

1.1 Introduction

The Virtual Closet Management System programme allows users to organise their closet items and put outfits together. The approach is designed to help users keep record of their components and bring trendy ensembles without having to go through their whole collection. The program me is composed of three microservices: Searching and Filtering, Outfit Creating, and Clothig Items. Users may search and filter their clothing items using the Search and Filter microservice, while the Outfit Creation microserrvice suggests outfit combintions depending on user preferrences. The Items service manages the apparel wardrobe.

The initiative aims to help clients makes better use of their clothing by tackling the issues associated with closet organisation. Fast fashion encourages consumers to buy more clothing than they need, resulting in cluttered closets and a less of organisation. Theis project solves this problem by allowing users to categorise their clothing and put together ensembles that suit their preferences.

The ASP.NET Core framework was used to develop the application, which features a microservice design and is separated into three distinct services. Each microservice is self-contained and was created to accomplish a specific task. Each microservice was designed to achieve a specific goal and may be expanded separately. Informatio is kept in a SQL Server Management Studio, and APIs are used to access it.

The apparels service offers APIs for curating, modifying, and removing clothing items. The microserviice also has APIs for retrieving all clothes or a particular piece of clothing. The Search and Filtering service APIs allow you to filter clothing by type, size, and color. Furthermore, the service provides APIs for searching for clothes goods using keywords. APIs for creating, modifying, and removing outfits are provided via the outfit creation microservice. APIs are provided by the microservice for obtaining all clothes or a specific fit. The service also includes APIs for making outfit combining suggestions based on user choices.

In conclusion, the project is a convenient and friendly solution to organize your clothing. The programmer supports users in clearing up clutter and increasing organization by offering a consolidated platform for tracking clothing items and putting together ensembles.

Using the Search and Filter microservice to browse apparel and find specific things is a quick and effective method to do it. By providing outfit ideas based on user choices, the Outfit Creation microservice delivers a one-of-a-kind and customized experience. The strategy encourages clients to use their clothes products more efficiently and avoid making additional purchases, which improves sustainability.

1.2 Problem Statement

People regularly struggle with deciding how to dress, forget what clothes they have, and buy unnecessary clothing items. Keeping track of all the items in your wardrobe may also be tough. These issues can cost time and money, as well as result in congested closets.

We propose creating a virtual wardrobe management system to address the issue.

- With this application, users will be able to create their wardrobe online, giving a simple efficient way to track their items.
- Users will be able to create items to their wardrobes, manage hem by category, type, and size, and make outfits from their existing apparel.
- The project will also make personalized style suggestions based on user choices, fashion and occasions.

1.3 Objectives

- **Effective wardrobe organization:** The project's primary purpose is to provide users with simple way to organize their stuff. Users may submit images of their apparel items and keep them in the system for easier access and management.
- **Personalization:** The system strives to provide a personalised experience by taking into account users' preferences, body types, and the events for which they require apparel.
- **E-commerce platform integration:** The project's goal is to connect with e-commerce platforms so that clients may buy items that complement their current wardrobe and receive recommendations for new purchases based on their choices and styling.
- **Data analysis:** The project seems to provide users with information on their clothing and fashion preferences by utilising data analytics. You may make informed decisions with the help of this knowledge.

1.4 Methodology

The project's goal is to use technologies to addressclothing management concerns and make styling recommendations. A variety of approaches are used to guarantee that system is efficacy, usable, and useful.

- **Collecting requirements**: The virtual wardrrobe management system's requirements were gathered early in its development. Understanding user requests, system charracteristics, and the technologies with which the system should be constructed were all necesssary.
- **System design:** The following phases were required for the system's design. This necessitated the creation of a system architecture capable of managing the projects's many demands, like generating outfit ideas.
- **System development:** The Agile approach was used. It was necessary to divide the devellopment proccess into shorter sprints or iterations to accomplish this.
- **Testing:** Unit testing was performed to ensure that each component of the system was working properly. Integrrity testing examined how various system components interacted with one another. User testting was conducted during acceptance testing to check that the system fulfilled thier needs and was simple to use.
- **Deployment:** The technology is based on a cloud platform. This included setting the platform to fit the system's needs as well as ensuring that the system was scalable and capable of withstanding growing traffic.
- **Maintenance:** Once implemented, the system required ongoing maintenance to ensure that it continued to function effectively. This necessitated monitoring for issues, resolving bugs, and introducing new functionality.

1.5 Organization

A project organisation is a framework that makes it easier to coordinate and carry out project tasks. Its major purpose is to facilitate relationships among team members with the least number of disruptions, overlaps, and conflict.

- **Planning phase:** The system planning offers a road map for project mannagers.

  From pre-plannning and stakeholder meettings through researrch, writing, scheduling, and final approvall, we've got you covered. All of these procedures and subtasks contribute to a successful project that is in line with the vision and final goals of the sponsor.
- **Analysis phase:** During the analysis phase, the project team will gather information on the virtual wardrobe management system's requirements and limits.

- **Design phase:** During this phase, the project team will use the requirements specification to develop the virtual wardrobe management system. A system architecture must be built as part of this, as must a database schema, a user interface, and the appropriate programming language and technology stack. The design process will result in a complete design document.
- **Implementation phase:** During this phase, the project team will build the virtual wardrobe management system using the design document as a guide. This includes writing code, testing the system, and correcting any problems. The implementation phase will result in a working system that meets all criteria and standards.
- **Testing and quality assurance phase:** During this phase, the project team will thorouughly test the virtual wardrobe management system to verify it meets specifications and performs as expected. This section includes inforrmation on system testing, integration testing, and unit testing. Two quality assurance duties that will be included in the testing process are performmance testing and code reviews.
- **Deployment and maintenance phase:** During this phase, the virtual wardrobe management system will be deployed to the production environment. The prroject team will support and educate users while keeping an eye out for any flaws or problems with the system. As needed, updates and bug fixes, as well as other mainntenance activities, will be performmed.

1.6 Solution Summary

1.6.1 Scope

The clothing items can be uploaded by users. In essence, the clothing item microservice will handle the user's things. The search and filter will handle the microservices' searching and filtering based on different factors like color, kind, size, and so on. The user authentication and authorization microservice ensures that a legitimate user logs in and registers. The system will also enable user authorisation to guaranttee that only authorised userrs can accesss and manage their wardrobe.

The project has an outfit creation tool, allowing users to mix and match their items to create new outfits. The algorithm will provide recommendations based on the user's tastes and the things in their closet. Users will be able to save and share data.

A model for tracking the frequency and use of clothing and ensembles will also be included in the gadget. Users will obtain a better understanding of their particular fashion preferences and be better prepared to make their next buy.

The web-based system will be accessible via desktop and mobile devices. It will feature an aesthetically beautiful and user-friendly modern interface.

The modules that must be produced as part of the project are listed below:

Requirement No.	Requirement Name	Requirement Description
REQ_01	Clothing item management	This module is a Middleware Microservice that accomplishes the following tasks: • The UI component would have the neccessary input fields as well as a user interface for uploading the clothing item. • The middleware component would be in charge of processing and validating the input data, which would compprise the item name, type, color, and image. • Finally, the service allows other sections of the programme to access the submitted data using a standardised API or interface.
REQ_02	Search and Filter management	• This is a Middleware service that does to filter clothing products by colour, size, fabric, and style. • Keeps a clothing item index for easier searching and filterinng.

REQ_03	Outfit Creation	This module is a Middleware Microservice that does the following: • The client requests that the outfit creaton service build an outfit for a certain occasion. • The outfit creation microservice requests a collection of clothiing items that match the occasion and intennded style from the clothing items microservvice. • The clothing items service sends the things required to the outfit creation microservice. • It then proceses the items and creates a suitable outfit and sends it back to the client.
REQ_04	User Authentication and Authorization	This service, which the app makes use of, should contain an authenticatin mechanism that wants people to sign up or log before utilizing the applicatin's features.
REQ_05	Virtual Wardrobe Management portal	An online plattform that allows users to login and perform the following operrations: • Use the proper credentals to access the portal. • Look through the wardrbe's apparel inventory. • Add a new peice of clothing to your closet. • Take a garment from your closet. • Look for and filter clothing items by color, size, type, and so on.

Table 1.6.1 Scope

1.6.2 Assumptions

- To gain access to the systemm, users must have a suitable device and an active internet connection.
- Users understand the fundamentals of using a web application.
- The system is not for commercial use; it is just for personal use.
- When users upload clothing photographs, the system expects that they will not break any copyright or intellecctual property rights, and thhat they will not use them for any unethical or unlawful activities.
- The system assumes that the data it hollds is secure and that it is often backed up to guard against theft or loss.
- The system anticipates that any flaws or problemms with the system will be addresssed and remedied as quickkly as feasible.

1.6.3 Dependencies

- **Hardware and software infrastructure:** In order to run the programmer, the system will require hardware such as servers, network components, and storage devices. Operating systems, web development tools, and database management systems are all required software dependencies.
- **Data management:** The system would require a robbust and secure database to store all essential data, such as user information, clothing items, outfit details, and preferences.
- **User authentication and authorisation:** To guaramntee that only approved users may use the system and its capabilities, the system would require user authentication and authorization.
- **Recommendation engine:** In order to enable the purchase of clothes or other related services, the system may need to interact with paymment processing.
- **Payment integration:** The system may need to interface with payment processing in order to facilitate the purchase of clothing or other relevannt services.
- **User interface design:** A user-friendly interface is esseential for users to interact with the system and its capabilities.

1.6.4 Risks

- **Technical complexity:** As part of the project, several microservices with sophisticated functions are being constructed. Technical complexities are a possibility, especially if the developers are unfamiliar with current trends and best practises. This might lead to delays, greater development costs, and quality issues.
- **Flaws in security:** The project requires the processsing of sensitivve user data, such as paymennt and login passwords. There is a possibility of security breaches if sufficient security measures are not established, which might lead to reputational damage and legal concerns.
- **Integration concerns:** Due to the project's reliance on a large number of microservices, integraation issues are conceivable. Every service must be able to communicate with the others without being interrupted, as failure to communicate might result in problems and downtime.
- **Concerns about scalability:** If the application is not designed to handle growing traffic and usage, scalability issues may occur as the user base grows. Performannce difficulties, crashes, and user dissatisfaction are possibilities.
- **Time and money constraints:** If the project is not adequately plannd and manage, it runs the risk of going over budget or missing deadlines. As a result, quality may suffer, the project's scope may be reduced, and other complications may arise.

1.6.5 Schematic Diagram

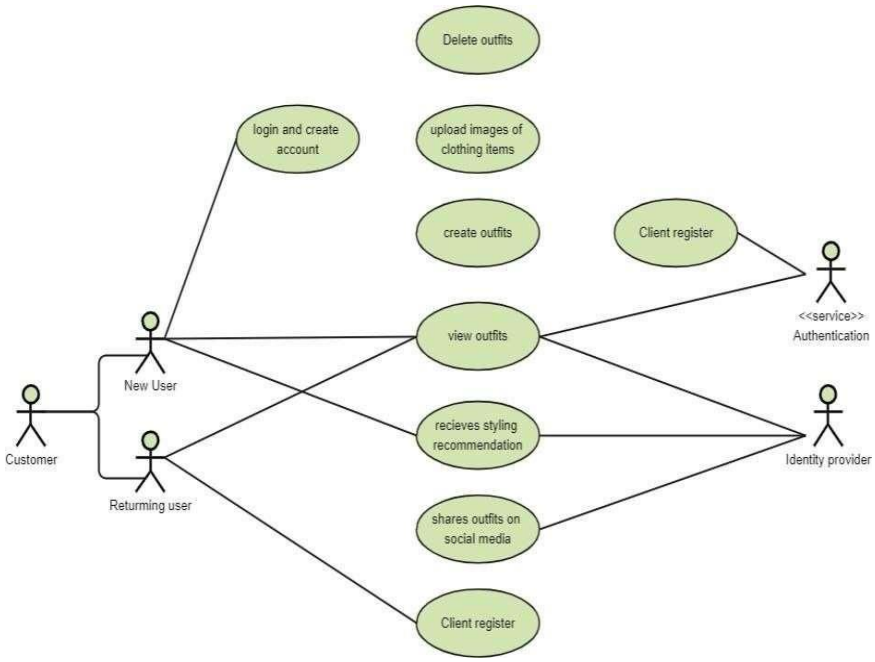

Figure 1.6.4 Schematic Diagram

CHAPTER-2 LITERATURE SURVEY

VWMS, a recent technical advancement, enables smooth administration of one's own clothing. VWMS has web-based interfaces and mobile aplications that allow customers to submit photographs of their clothing items as well as importannt information such as style and color for optimal management. Users don't have to worry abbout getting ready for big events when they utilize this strategy, which allows them to blend items from their virtual wardrobe.

Customers have quickly embraced virtual wardrobe management solutions as a result of their creation to keep their wardrobes organized and practical. Virtual wardrobe management systems have been the subject of several research, which have identified both their benefits and drawbacks.

By allowing users to quickly identify certain clothing items from their closet and create new ensembles without having to try them on physically, virtual wardrobe management solutions make it easier to organise and arrange apparel.

Virtual wardrobe organisation software improves users' capacity to make the most of already-owned apparel by recommending creative outfit combinations. This can encourage people to wear clothing more often, assist them in getting morre usage out of their clothes, and lesssen the need to frequently buy new thinngs.

On the other hand, there are substantial disadvantages to using virtual wardrobe management solutions. Accurately identifying the type of clothes and other factors like colour and size presents one of the system's major hurdles. The effectiveness of the system depends well the picture recognitin algorithms work, which might have limitations in some situations, including when clothing has similar colours or patternss.

The security and privacy of user data is another issue that virtual wardrobe management systems need to solve. There is a chance that the photos will be viewed or used inappropriately because the system requires users to submit photographss. System, to sum up, are a potential technology that helps users to manage their clothing items more susccessfully, promote sustainable fashion practises, and encourage users to get more use out of their items. There are a few drawbacks to this technology, too, including issues with the system's precision in identifying garment features and the security and privacy of user data.

Challenges:

- **Data accuracy:** Updating the board's virtual closet with accurate data is one of the most difficult aspects of the system. Making sure the clothing items are appropriately listed, categorised, and represented is part of this.
- **Integration of frameworks:** Virtual closet the board arrangements should be compatible with a variety of frameworks, including online company platforms, stock administration frameworks, and personal styling services. Broad planning and involvement are needed to enable smooth coordination.
- **Protection and security:** The executives' virtual filing cabinets should protect the client information. This includes protecting against information leaks and ensuring that customer data isn't misused or handled carelessly.
- **Client courtesy:** The executive's arrangements should be simple to use and provide clients with a solid advantage recommendation in order to energise acknowledgement.
- **Adaptability:** The framework should have the ability to scale as the number of clients and items it handles grows. This will allow it to handle the increased workload.

Technologies And Solutions:

- **Picture acknowledgment and arrangement:** This invention makes use of PC vision computations to classify clothing items based on visual characteristics like tone, surface, pattern, etc. The three most well-known photo ID and design innovations for clothing items are Google Cloud Vision, Amazon ID, and Microsoft Sky Blue Mental Administrations.
- **Regular language processing (NLP):** This invention uses AI algorithms to understand and respond to inquiries made in everyday language by customers. The application of NLP for search and concept might be accomplished by looking at literary representations of clothing products. Three notable NLP developments for clothing include IBM Watson, Google Cloud Normal Language, and Amazon Fathom.
- **Cooperative separating**: This innovation prescribes products to clients in light of them earlier inclinations and conduct.

S.No.	Authors	Published By	Methodology	Database	Results	Limitations
1	Zhang et al. (2018) [1]	Journal of Fashion Technology & Textile Engineering	Developed a virtual wardrobe system utilizing image recognition and recommendation algorithms	Fashion dataset with labeled clothing items	Improved outfit suggestions based on user preferences and fashion trends	Limited dataset for training the recommendation algorithm
2	Li et al. (2019) [2]	International Journal of Human-Computer Studies	Developed a mobile application for virtual mixing and matching of clothes	Limited availability of real-time fashion advice based on current trends	Users were able to experiment with different styles and receive real-time fashion advice	Limited availability of real-time fashion advice based on current trends
3	Chen et al. (2020) [3]	IEEE Access	Proposed a deep learning-based approach for clothing item recognition and categorization	Limited to recognizing and categorizing a predefined set of clothing items	Achieved accurate clothing item identification and classification	Limited to recognizing and categorizing a predefined set of clothing items
4	Xie et al. (2021) [4]	ACM Transactions on Multimedia Computing,	Developed a recommendation algorithm based on	Limited to the availability of user preference	Improved outfit recommendations based on user	Limited to the availability of user preference and

		Communications, and Applications	collaborative filtering for suggesting outfit combinations	and fashion trend data	preferences and fashion trends	fashion trend data
5	Liu et al. (2020) [5]	Proceedings of the IEEE International Conference on Multimedia and Expo	Combined AR and 3D garment modeling for a virtual dressing room system	Limited accuracy in mapping clothes onto the user's body using computer vision techniques	Users could virtually try on clothes and visualize them using AR technology	Limited accuracy in mapping clothes onto the user's body using computer vision techniques
6	Wang et al. (2020) [6]	International Journal of Information Management	Developed a mobile application with cloud-based storage and synchronization capabilities	Limited to the availability and reliability of cloud-based storage and synchronization	Enabled users to manage their wardrobes on-the-go and share outfit ideas	Limited to the availability and reliability of cloud-based storage and synchronization
7	Tang et al. (2019) [7]	Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies	Proposed a gesture-based interaction system for controlling the virtual wardrobe	Restricted to the appropriate and trust of gesture recognition algorithms	Provided a more intuitive and immersive user experience	Limited to the accuracy and reliability of gesture recognition algorithms
8	Kim et al.	Computers in	Investigated user	Limited sample	Users found the	Limited sample size

	(2022) [8]	Human Behavior	acceptance and satisfaction of a virtual wardrobe management system	size and potential biases in user feedback	system helpful in organizing and planning outfits, leading to increased satisfaction	and potential biases in user feedback

Table 2.1 Early studies on virtual wardrobe management and recommendation systems

CHAPTER-3 SYSTEM ANALYSIS AND DEVELOPMENT

3.1 Architecture And Design

The several levels and elements of the system, their relationships, and how they were created to satisfy the project's specifications will all be covered in this part.

Architecture

Individual services can be launched independently thanks to the system's use of a microservices architecture. Every microservice manages a certain functionality and interacts with other services via APIs. This technique has the benefits of scalability, flexibility, and resilience. The Virtual Wardrobe Management System comprises four main microservices:

- Microservices for managing clothing items
- Creating outfits
- Searching for information
- Managing users are also available.

These microservices all interact with one another using APIs and are each in charge of a certain functionality. The database for clothing items is maintained by the Clothing Items Management Microservice. Creating and managing outfits is the responsibility of the outfit creation microservice. The apparel items are searched for and filtered by the Search and Filter Microservice. The management of user accounts and authentication is handled by the User Management Microservice.

Design

Four primary spaces that each correspond to a different microservice have been created within the framework. The areas are called Clothing Things, Outfit Creation, Search and Channel, and Client The executives. Every microservice's provided APIs control the communications between these regions.

The executives and the outfit plan are managed by outfit creation. It provides APIs for adding and removing garment parts from outfits, as well as for delivering and erasing outfits. Locating and classifying clothing items is supervised by the Hunt and Channel area. It offers APIs for finding and sorting items according to a variety of criteria, including tone, size, and type. Conclusively, the Client The executives space takes care of client confirmation and records. It provides APIs for

creating and managing customer accounts as well as for verification and authorization.

Last but not least, the Virtual Closet's engineering and plan. The Board Framework uses a microservices approach, which promotes adaptability, flexibility, and adaptability. Four fundamental areas of the framework, each of which is connected to a different microservice, have been separated. Each microservice is delivered as a distinct containerized programme, and communication between them is done by calm APIs.

System Architecture Diagram

Following is a diagram of the Virtual Wardrobe Management System's architecture and layout:

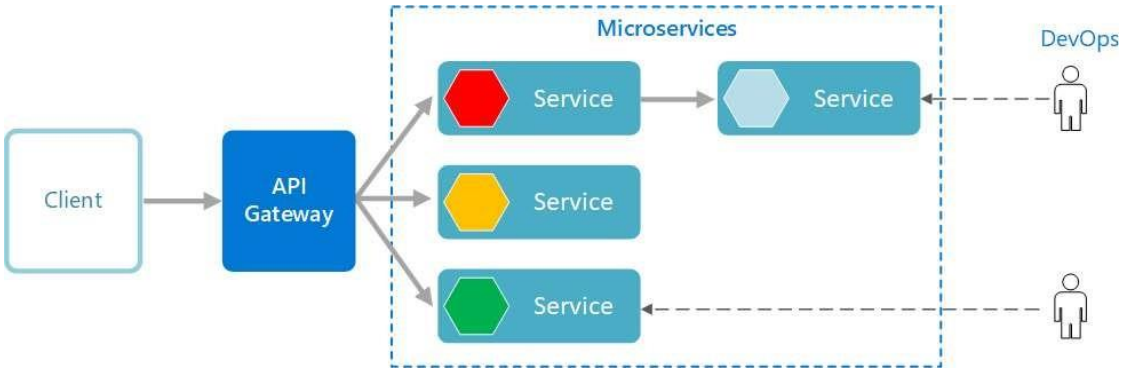

Figure 3.1 System Architecture Design

The system's various parts and layers are depicted in the diagram, along with their relationships and how each was created to satisfy the project's needs. As distinct containers, the microservices are displayed.

3.1.1 List of Components

1. Frontend/Ul:

- HTML/CSS/JavaScript
- A library or front-end framework (such as React, Angular, or Vue.js).
- Software for creating user interfaces (such as Adobe XD and Sketch).

2. Backend/API:

- The C#.NET Framework

- Developing RESTful APIs with ASP.NET Core
- Entitty Framework Core for database administration
- Swagger is used for API documentation.

3. Database:

- Microsoft SQL erver or a different RDBMS (relational database management system).

4. Additionally

The following advantageous for deployment and development:

- Version control using Git/GitHub
- Visual Studio for programme development.

3.2 Algorithm

The virtual wardrobe management system takes the following algorithms into account when curating:

1. The Clothing Item Addition Algorithm:

- Request information about the item of clothing, such as the name, colour, size, type, etc. from the user.
- Verify the information provided and store the article of clothing in the database.
- Show a message to show that you've successfully attached the piece of clothing.

2. An algorithm for finding clothing:

- Request the user's input for the search criteria, which may include name, colour, size, kind, etc.
- On the basis of the search parameters, retrieve the pertinent apparel items from the database.
- Give the user a view of the clothing pieces.

3. Algorithm for the creation of an outfit:

- Request that the user choose the apparel to be worn in the ensemble.
- Confirm the clothes you've chosen are appropriate by comparing their colours, styles, etc.
- To make the ensemble, generate a random combination of the chosen clothing components.
- Show the user what is wearing.

4. Making clothing recommendations using an Algorithm

- Retrieve and retrieve the clothing items from the user's previously generated ensembles.
- Examine the apparel to establish the user's fashion preferences.
- According to the user's preferred styles, retrieve comparable apparel products from the database.
- Utilising the obtained clothing articles, create outfit combinations.
- Using the user's choices and input, recommend certain outfit combinations to them.

3.3 Analytical Development

Breaking down data to identify examples and patterns that can help decision-makers and project managers better understand execution is a crucial part of logical project improvement. There are several areas where scientific advancement can be used to a virtual closet-the-board framework:

- **An assessment of client behaviour:** This entails dissecting client behaviour to identify examples and trends in how clients interact with the framework. For instance, what materials are used the most frequently, what ensembles are popular, and what categories of clothing are typically added to customers' wardrobes the most frequently. By adjusting the framework in light of client behaviour, this information can be used to improve the client experience even further.
- **Deals investigation:** On the off chance that the virtual closet the board system has an online business component, deals investigation might provide crucial information about customer behaviour. Examining the apparel items that are most frequently purchased, those that are left in the shopping cart, and those that provide the highest earnings are some examples of this. By using this information, stock management and valuation processes can be strengthened.
- **Investigation of patterns:** The framework will be able to recommend outfits more effectively if style is examined. Among other things, this can entail looking at popular clothing brands, fashion blogs, and internet entertainment trends. Additionally, the system might make use of image recognition technology to identify patterns in the clothes that customers have saved and suggest similar items.
- **Stock analysis:** The system might use stock information to analyse patterns in how clothes items are worn and how frequently they are replaced. To improve stock management and valuation systems, use this data.

- **Examining personalization:** It is important since it strengthens the structure of a virtual workplace. To help the framework provide more precise suggestions and improve the overall client experience, client information can be broken down into other categories, such as apparel preferences, body type, and variety preferences.

In order to apply scientific improvement in a virtual closet the exectutives framework, information should be gathered and saved in a data collection. Analytical improvement is an iterative process that comprises designing, planning, executing, and fine-tuning a soln to a given problem. The primary purpose of logical developmnt is to discern between insights and knowledge and information that can be utilizeed to support inforrmed decisions and enhhance business results.

Scientific advancements could be made in the virtual closet the board framework in the following ways:

- **Defining the problem**: This includes finding the specific problem or challenge that the virtual wardrobe for executives framework is meant to solve. If customers have trouble organising and keeping track of their clothing items, for instance, that might be the problem.
- **Information gathering:** This innvolves identifying the sourcces of information that will be used to build the virtuall closet the board architecture. Information from clothing merchants, user-generated content, and information from other sources could all be included.
- **Information readiness:** This entails organising, updating, and cleaning up the information to make sure that it may be successfully broken down inside an organisation.
- **Information examination:** which entails the extraction of experiences and information from the information using quantifiable and AI techniques. For instance, bunching calculations might be used to compile comparable clothing items, and relapse anallysis could be used to predict customer preferences based on their prior interactions with the system.
- **Implementation:** This entails creating and sending the virtual closet the board structure based on knowledge and data gleaned from the information inspection.
- **Examining and improving:** Examining the virtual closet the executives framework's viability as well as improving it in response to feedback from clients and other partners are some of the things that are covered in this.

Scientific advancement generally plays a crucial role in the success of the virtual closet the board framework by enabling the framework to provide clients with personalised recommendations and information based on their unique preferences and ways of acting.

3.3.1 Data Model

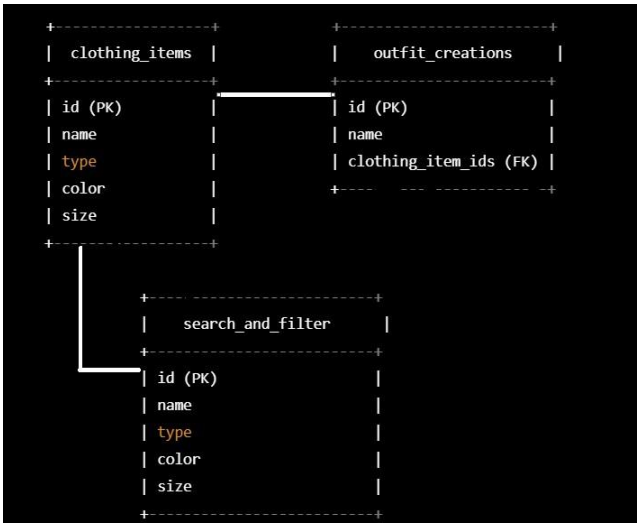

Figure 3.3.1 Data Model

3.3.2 Table Structure

Tables Structure for services:

1. Clothing Item Management

Column Name	Data Type	Length	Nullable
Id	int	-	NO
Name	varchar	100	NO
Type	varchar	50	NO
Color	varchar	50	NO
Size	varchar	10	NO
image_url	varchar	255	YES

Table 3.3.2(1) Clothing item microservice table structre

2. Search And Filter

Column Name	Data Type	Length	Nullable
id	int	-	NO
Name	varchar	100	NO
type	varchar	50	NO
color	varchar	50	NO
size	varchar	10	NO
image_url	varchar	255	YES

Table 3.3.2(2) Search and Filtering microservice table structure

Clothing items, Search and Filter, and the database they share have the same table structure.

3. Outfit Creation

Column Name	Data Type	Length	Nullable
Id	int	N/A	NO
Name	varchar	50	NO
ImageUrl	varchar	500	NO
clothing_item	int	N/A	NO

` Table 3.3.2(3) Search and Filter microservice table structure

3.4 Project Development

Users may manage their clothing, make outfits, search and filter through their closets using the web-based Virtual Wardrobe Management System. Users should be able to keep track of their belongings and put together attractive outfits with the help of this technology.

A microservices architecture was used to build the project, and it contained four distinct microservices: User, Clothing Items, Outfit Creation, and Search and Filtering. Increased flexibility and scalability were achieved as a result of the independent creation of each microservice and its interaction with the others via APIs.

User registration, login, and authentication are all under the control of the User microservice. Furthermore, it manages user-specific information like profiles and preferences.

The user's clothing items are managed by the Clothing Items microservice, which is also responsible for item deletion and information updates. Each object can be categorised by kind, colour, and other characteristics and is linked to a user.

By selecting items from their wardrobe, users can create outfits using the Outfit Creation microservice. For later viewing, users can create and store multiple costumes. The software includes a social component that allows users to exchange outfits with one another.

3.4.1 Clothing Item Microservice

The Virtual Wardrobe Management System's clothing items are managed by the Clothing Item microservice through CRUD (Create, Read, Update, Delete) actions.

Development of Clothing Item Management Microservice:

The Entity Framework Code First method is used by the Clothing Item Microservice to communicate with the database and was constructed using the ASP.NET Core Web API. The Clothing Item model includes attributes like Name, Description, Image, Categorry, SubCategories, ype, Size, and UserId.
There are three main parts to the clothing item microservice:

- ClothingsItem_Model
- ClothingsItem_Controller
- ClothingsItem_Repository

Clothing Item Repository:

In order to communicate with the database using Entity Framework, the Clothing Item Repository is in charge of doing so. For CRUD actions on the Clothing Item entity, it contains methods.

A definition of the Clothing Item Repository can be found in the "ClothingItemRepository.cs" file, which is kept in the project's "Repositories" folder.

The Generic Repository, a personalised repository with generic methods for carrying out CRUD operations on any entity type, is the custom repository from which the Clothing Item Repository derives. In order to carry out the particular activities necessary for the Clothing Item entity, the Clothing Item Repository overrides the generic repository methods.

By means of the User ID foreign key field, the Clothing Item Microservice is linked to the User Microservice. When searching for and filtering data pertaining to the Clothing Item entity, the Search and Filtering Microservice and the Clothing Item Microservice collaborate.

Finally, the Clothes Item Microservice is composed of a Model, a Contrroller, and a Reppository and was developed utilising the Entity Framework Code First technique. HTTP responses, and the Repository communicates with the database and performs CRUD operations on the Clothing Item entity using object Framework. The Clothing Item Microservice is linked to the User Microservice and the Search and Filtering service in order to carry out certain tasks.

3.4.2 Outfit Creation Microservice

For the Virtual Wardrobe Management System, outfit creation is handled by the Outfit Creation microservice. It adheres to a conventional ASP.NET Web API's standard architecture. To perforn CRUD operations on the system or the application, use the Outfit Creation microservice.
Users will have a thorough experience using it because it is made to integrate easily with the other microservices in the system.

3.4.3 Search And Filtering Microservice

The Virtual Wardrobe Management System's Search and Filtering microservice is in charge of supplying a way to search for and filter out clothing items according to different criteria.
The four-file microservice, which was created using Entity Framework, is composed of the following four files:

- SearchAndFilterController.cs
- ClothingItem.cs
- SearchAndFilterRepository.cs

- 'ISearchAndFilterRepository.cs

Using filters like colour, kind, id, and name of the item or outfit, this microservice will enable the user to locate the clothing items or outfits he has produced.

3.4.4 User Authentication And Authorization Microservice

In the virtual wardrobe management system, user authentication and authorization are managed by the microservice for user authentication and authorization. Password hashes are one-way functions that take a password for input and produce a fixed-length string of characters for output. The password hash represents this output. Instead of storing the real password, the database stores the password hash. When a user logs in, the system hashes their password, compares it to one that has already been saved, and accepts the hashed password as input. The hashes must match for user authentication to take place. Adding a random string of characters before hashing a password is known as adding a "salt" to it. When an account is created, which is specific to user, is formed. Using precomputed hash tables to crack the password is made impossible by the salt. It becomes harder for attackers to break many passwords at once by adding a distinct salt to each password, making the hash unique as well.

The PasswordHash and PasswordSalt attributes are utilised in the virtual wardrobe management system's user permission and authentication microservice to store the hashed and salted passwords for each user. The password is hashed and salted when a user registers or modifies it, and then it is saved in the database.

The functionality for this microservice's registration, login, and password update is handled by the user controller, while the UserRepository and IUserRepository interfaces specify the ways to communicate with the user database.

CHAPTER-4 PERFORMANCE ANALYSIS

4.1 ASP .NET Core

The open-source, crosss-platform ASP.NET Core web frammework was created by Microsoft to give programmers the tools they need to construct cutting-edge, cloudd-based, and network-connectted apps.

It uses async/await, quick garbage collection, and lightweighht processing technologies to handle heavy traffic and big applications. In addition, ASP.NET Core enables straightforward cloud integration and may be deployyed on a variety of hostinng platforms, including Microssoft Azure, AWS, and Docker containers.

All in all, ASP.NET Core is a modern, flexible, andd reliable online frammework that enables developers to create cutting-edge, cloud-native apps using their own programmming languagge.

4.2 Entity Framework

.NET's Entity Framework (EF) is a well-known object-relationa mapping (ORM) frameworrk. The set of packages makes it possible for programmers to use.NET objects to connect to databases. Database tables could be linked to CLR (Common Language Runtime) objects and vice versa using EF to simplify data interaction for developers. SQL Server, MySQL, Oracle, and PostgreSQL are just a few of the database management systems that EF supports. Database-first, model-first, and code-first data access scenarios are among those that are supported.

Entity Framework enables us to communiicate with a databse using an object-orientd programming approach. We're using Entity Framework Core, a simplified, cross-platform version of Entity Framewrk that can connect to a number of different databass. We use Entity Framework to link our data models to database tables; these models are expressed as classes in C# and include concepts like User, Outfit, ad Clothing Items.

Additionally, a DbContext class (ApplicationDbContext) was created to define a database session. Our data models all have DbSet characteristics that let us query the pertinent database tables.

The UserRepository class makes use of Entity Framework in order to create, read, update, and delete User objects from the database. We also utilise Entity Framework for password salt and hash validation when a user logs in. Overall, EF simplifies database interactions by enabling us to wrk with C# objects rather than raw SQL, and it does so by removing a significant amount of the boiler plate code for us.

4.2.1 Database First Approach

Databse-first, model-fist, and code-first interactions with the EF Object-Relational Mapping (ORM) framework are all possible. When using the database-first methodology, Entity Framework generates entity clases basd on the existing database schema.

Create a new project in Visual Studio to begin connecting to the database, and then ad a new item of type ADO.NET Entity Data Model to the poject. When creating an entity data model from an existing database, select the Database option in the Entity Data Model Wizard. After the wizard has read the database schema, the tables, views, stored procdures, and relatioships will be used to build an entity data model (EDM).

It might be time-consuing and error-prone to regenerate the entity clases after making modifications to the database structure. Additionally, you may need to manually modify them if the Entity Framework entity classes don't always meet your needs.

The Database-First strategy provides a strong and flexible way to use Entity Framework while working with current or old databases, to sum up. Prior to choosing to apply this methodology to your project, it is critical to evaluate both the advantages and disadvantages of the approach.

4.2.2 Code First Approach

You can build a database schema from your program's classes using Entity Framework's (EF) Code First methodology.

In order to represent the database context of the application while using Code First, you typically start by developing a DbContext class. The properties of this class's DbSet correspond to the collections of entities in your database.

As your code is modified, you can then update the database schema using Entity Framework's migration feature.

The Code First approach includes the following key elements, among others:

- Classes defined as POCOs: You do not need to use EF to define your classes as POCOs. Your code is therefore easy to test and integrate with different frameworks and technologies.
- Mapping based on conventions: Entity Framework maps your POCO classes to the databse structure based on conventions. For example, Enitty Framework would presume that the related databse table is called "Customers" and has a primary key colmn named "CustomerId" if your class is called "Customer" and has a field named "CustomerId".
- Explicitly configuring the database schema is possible with the help of the fluent API. Now that this has happened, you have more control over the database.
- Migrations: The EF migrations feature allows you to alter the database schema as your code is modified. With no data being lost, you can now change the schema.

The name, description, and image of the outfit are just a few examples of the attributes that the Outfit class contains that reflect outfit information. Similar to this, the clothing item depicts information about the user's clothing item, and the search and filter microservice is utilised to search for clothng item or outfits based on a certain parametesr like type, color, kind, etc. The password hash and salt can both be stored in attributes of both classes.

The domain model is connected to the database schema using EF, which uses a set of standards and traits. When EF maps attributes and classes to database rows and colmns, it does so according to conventions. As an illustraton, EF uses the class name to choose the name of the database table and the property name to choose the name of the column. In addition to standards, EF offers attributes that developers can use to tailor the mapping behavior.

4.2.3 REST API

RESTful APIs are used in this project with the goal of giving the different microservices a standardised way to connect with one another. Each microservice exposes its functionality through a set of REST API endpoints, that client-side applications and other microservices can use to contact it.

In this systemt's implementation of the REST API endpoints, activities on the resources available by each microservice are performed using the HTTP protocol and the HTTP methods (GET, POST, PUT, DELETE, etc.). The API endpoints return data in a format called JSON, which is small and widely used for data exchange

The Search and Filter microservice exposes endpoints for discovering and sorting clothes based on a variety of criteria.

The client-side application can make requests to these API endpoints in order to interact with the microservices and complete a variety of tasks like browsng and filterig clothing items, creating new outfits, and managing user accounts. This enables the development of a decoupled, modular architecture that permits the development, launching, and scaling of individual microservices without affectng other system elements.

4.2.4 Source Code

1. Clothing Item Microservice

Entity: ClothingsItem.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer.Models
{
    14 references
    public class ClothingItem
    {
        2 references
        public string Name { get; set; } = null!;
        2 references
        public string Type { get; set; } = null!;
        2 references
        public string Colour { get; set; } = null!;
        2 references
        public string Size { get; set; } = null!;
        [Key]
        5 references
        public int Id { get; set; }
        2 references
        public string ImageUrl { get; set; } = null!;
        //public int UserId { get;set; }

    }
}
```

Figure 4.1 Entity: ClothingsItem.cs

Interface: IClothingRepositry.cs

```
using DataAccessLayer.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer.Repositories
{
    3 references
    public interface IClothingRepository
    {
        2 references
        void AddClothingItem(ClothingItem item);
        2 references
        void DeleteClothingItem(int Id);
        2 references
        IEnumerable<ClothingItem> GetAllClothingItems();
        4 references
        ClothingItem GetById(int Id);
        2 references
        void UpdateClothingItem(int id, ClothingItem item);
```

Figure 4.2 Interface: IClothingItem.cs

Repository: ClothingRepository.cs

```
public void DeleteClothingItem(int Id)
{
    var ClothingItem = context.ClothingItems.Find(Id);
    if (ClothingItem != null)
    {
        context.ClothingItems.Remove(ClothingItem);
        context.SaveChanges();
    }
}

4 references
public ClothingItem GetById(int Id)
{
    var getbyid= context.ClothingItems.FirstOrDefault(x => x.Id == Id);
    if(getbyid == null)
    {
        throw new ArgumentException("no clothingitem found with this id");
    }
    return getbyid;
}
```

Figure 4.3 Repository: ClothingsRepository.cs

```
public class ClothingRepository : IClothingRepository
{
    private readonly ApplicationDbContext context;
    0 references
    public ClothingRepository(ApplicationDbContext context)
    {
        this.context = context;
    }
    2 references
    public void AddClothingItem(ClothingItem item)
    {
        //var isData = context.ClothingItems.Where(x => x.Id ==
        //if(isData)
        //{
        //}
        context.ClothingItems.Add(item);
        context.SaveChanges();
    }
    2 references
    public IEnumerable<ClothingItem> GetAllClothingItems()
    {
        return context.ClothingItems;
    }
```

Figure 4.4 Repository: ClothingsRepository.cs

```
2 references
public void UpdateClothingItem(int id,ClothingItem item)
{
    var result = context.ClothingItems.FirstOrDefault(x => x.Id == id);
    if(result == null)
    {
        throw new ArgumentException("Id not found");
    }
    result.Name = item.Name;
    result.Type = item.Type;
    result.Colour = item.Colour;
    result.Size = item.Size;
    result.ImageUrl = item.ImageUrl;
    context.SaveChanges();

}
```

Figure 4.5 Repository: ClothingsRepository.cs

```
[ApiController]
[Route("/api/[controller]")]
1 reference
public class ClothingItemsController : Controller
{
    private readonly IClothingRepository _dbContext;
    0 references
    public ClothingItemsController(IClothingRepository dbContext)
    {
        _dbContext = dbContext;
    }
    [HttpGet]
    0 references
    public IEnumerable<ClothingItem> GetAllClothingItems()
    {
        return _dbContext.GetAllClothingItems();
    }

    [HttpGet("{id}")]
    1 reference
    public IActionResult GetById(int id)
    {
        var item = _dbContext.GetById(id);
        if (item == null)
        {
            return NotFound();
        }
        return Ok(item);
    }
```

Figure 4.6 Repository: ClothinsgRepository.cs

Controller: ClothingItemsController.cs

```
[HttpPost]
0 references
public IActionResult AddClothingItem(ClothingItem item)
{
    _dbContext.AddClothingItem(item);
    return CreatedAtAction(nameof(GetById), new {id=item.Id},item);
}

[HttpPut("{id}")]

0 references
public IActionResult UpdateClothingItem(int id, ClothingItem item)
{
    if (id != item.Id)
    {
        return BadRequest();
    }

    var existingOutfit = _dbContext.GetById(id);
    if (existingOutfit == null)
    {
        return NotFound();
    }
    _dbContext.UpdateClothingItem(id, item);

    return NoContent();
}
```

Figure 4.7 controller: ClothingsItemController.cs

```
[HttpDelete("{id}")]
0 references
public IActionResult Delete(int id)
{
    var existingOutfit = _dbContext.GetById(id);
    if (existingOutfit == null)
    {
        return NotFound();
    }

    _dbContext.DeleteClothingItem(id);

    return NoContent();
}
```

Figure 4.8 controller: ClothingsItemController.cs

2. Outfit Microservice

Entity: outfits.cs

```
14 references
public class Outfit
{
    [Key]
    4 references
    public int OutfitId { get; set; }
    [Required]
    2 references
    public string OutfitName { get; set; } = null!;
    [Required]
    0 references
    public virtual ICollection<ClothingItem> Items { get; set; } = null!;
    2 references
    public string OutfitImageUrl { get; set; } = null!;
    [Required]
    0 references
    public DateTime CreatedAt { get; set; }
    [Required]
    1 reference
    public DateTime UpdatedAt { get; set;}
    0 references
    public Users Users { get; internal set; } = null!;
}
```

Figure 4.9 Entity: outfitss.cs

Interface: IOutfitRepositry.cs

```
3 references
public interface IOutfitRepository
{

    2 references
    void CreateOutfit(Outfit outfit);
    2 references
    IEnumerable<Outfit> GetOutfits();
    4 references
    Outfit GetOutfitById(int id);
    2 references
    void UpdateOutfit(int id, Outfit outfit);
    2 references
    void DeleteOutfit(int id);

}
```

Figure 4.10 Interface: IOutfitsRepository.cs

Repository: OutfitRepository.cs

```
public class OutfitRepository:IOutfitRepository
{
    private readonly ApplicationDbContext context;
    0 references
    public OutfitRepository(ApplicationDbContext context)
    {
        this.context = context;
    }
    2 references
    public void CreateOutfit(Outfit outfit)
    {
        context.Outfits.Add(outfit);
        context.SaveChanges();
    }
    2 references
    public IEnumerable<Outfit> GetOutfits()
    {
        return context.Outfits.ToList();
    }
```

Figure 4.11 Repository: IOutfitRepository.cs

```
public Outfit GetOutfitById(int id)
{
    var result= context.Outfits.FirstOrDefault(c => c.OutfitId == id);
    if(result == null)
    {
        throw new ArgumentException("Id does not exist");
    }
    return result;
}
2 references
public void UpdateOutfit(int id, Outfit outfit)
{
    var outfitResult=context.Outfits.FirstOrDefault(c=>c.OutfitId== id);
    if (outfitResult != null)
    {
        outfitResult.OutfitName = outfit.OutfitName;
        outfitResult.OutfitImageUrl = outfit.OutfitImageUrl;
        outfitResult.UpdatedAt = DateTime.Now;
        context.SaveChanges();
    }
}
```

Figure 4.12 Repository: IOutfitRepository.cs

```
2 references
public void DeleteOutfit(int OutfitId)
{
    var outfit = context.Outfits.Find(OutfitId);
    if(outfit!= null)
    {
        context.Outfits.Remove(outfit);
        context.SaveChanges();
    }
}
```

Figure 4.13 Repository: IOutfitRepository.cs

Controller: OutfitController.cs

```
[ApiController]
[Route("/api/[controller]")]
1 reference
public class OutfitController : Controller
{
    private readonly IOutfitRepository _context;

    0 references
    public OutfitController(IOutfitRepository context)
    {
        _context = context;
    }

    [HttpGet]
    0 references
    public ActionResult<IEnumerable<Outfit>> GetOutfits()
    {
        var outfits = _context.GetOutfits();
        return Ok(outfits);
        //return _context.Outfits.ToList();
    }
```

Figure 4.14 Controller: OutfitController.cs

```
[HttpGet("{id}")]
1 reference
public ActionResult<Outfit> GetOutfitById(int id)
{
    var outfit = _context.GetOutfitById(id);

    if(outfit == null)
    {
        return NotFound();
    }
    return Ok(outfit);
}

[HttpPost]
0 references
public ActionResult<Outfit> CreateOutfit(Outfit outfit)
{
    _context.CreateOutfit(outfit);
    return CreatedAtAction(nameof(GetOutfitById), new { id = outfit.OutfitId }, outfit);

}
```

Figure 4.15 Controller: OutfitController.cs

```
[HttpPut("{id}")]
0 references
public IActionResult UpdateOutfit(int id, Outfit outfit)
{
    if(id!= outfit.OutfitId)
    {
        return BadRequest();
    }

    var existingOutfit = _context.GetOutfitById(id);
    if(existingOutfit == null)
    {
        return NotFound();
    }
    _context.UpdateOutfit(id,outfit);

    return NoContent();
}

[HttpDelete("{id}")]
0 references
public IActionResult DeleteOutfit(int id)
{
    var existingOutfit = _context.GetOutfitById(id);
    if(existingOutfit == null)
    {
        return NotFound();
    }
```

Figure 4.16 Controller: OutfitController.cs

```
    _context.DeleteOutfit(id);

    return NoContent();
}
```

Figure 4.17 Controller: OutfitController.cs

3. Search and Filtering Microservice

Entity: SearchAndFilter.cs

```
13 references
public class SearchAndFilter
{
    0 references
    public string Name { get; set; } = null!;
    2 references
    public string Type { get; set; } = null!;
    1 reference
    public string Colour { get; set; } = null!;
    1 reference
    public string Size { get; set; } = null!;
    0 references
    public string OutfitName { get; set; } = null!;
    0 references
    public int OutfitIfd { get; set; }
    [Key]
    0 references
    public int Id { get; set; }

}
```

Figure 4.18 Entity: SearchAndFilter.cs

Interface: ISearchAndFilterRepository.cs

```
3 references
public interface ISearchFilterRepository
{
    2 references
    List<SearchAndFilter> SearchByColor(string color);
    2 references
    List<SearchAndFilter> SearchByType(string type);
    2 references
    List<SearchAndFilter> FilterBySize(string size);
    2 references
    List<SearchAndFilter> FilterByType(string type);
}
```

Figure 4.19 Interface: ISearchAndFilterRepository.cs

Repository: SearchAndFilterRepository.cs

```
private readonly ApplicationDbContext _context;
0 references
public SearchFilterRepository(ApplicationDbContext context)
{
    _context = context;
}
2 references
public List<SearchAndFilter> SearchByColor(string color)
{
    return _context.SearchandFilter.Where(c => c.Colour == color).ToList();
}
2 references
public List<SearchAndFilter> SearchByType(string type)
{
    return _context.SearchandFilter.Where(c => c.Type == type).ToList();
}
2 references
public List<SearchAndFilter> FilterBySize(string size)
{
    return _context.SearchandFilter.Where(c => c.Size.Contains(size)).ToList();
}
2 references
public List<SearchAndFilter> FilterByType(string type)
{
    return _context.SearchandFilter.Where(c => c.Type.Contains(type)).ToList();
}
```

Figure 4.20 Repository: SearchAndFilterRepository.cs

Controller: SearchAndFilterController.cs

```
public class SearchAndFilterController : Controller
{
    private readonly ISearchFilterRepository _context;

    0 references
    public SearchAndFilterController(ISearchFilterRepository context)
    {
        _context = context;
    }

    [HttpGet("seacrh-by-color/{color}")]
    0 references
    public ActionResult<List<SearchAndFilter>> SearchByColor(string color)
    {
        var result= _context.SearchByColor(color);
        if (result == null)
        {
            return NotFound();
        }
        return Ok(result);
    }
```

Figure 4.21 Controller: SearchAndFilterController.cs

```
[HttpGet("seacrh-by-type/{type}")]
0 references
public ActionResult<List<SearchAndFilter>> SearchByType(string type)
{
    var result= _context.SearchByType(type);
    if(result == null)
    {
        return NotFound();
    }
    return Ok(result);
}
[HttpGet("filter-by-size/{size}")]
0 references
public ActionResult<List<SearchAndFilter>> FilterBySize(string size)
{
    var result= _context.FilterBySize(size);
    if(result == null)
    {
        return NotFound();
    }
    return Ok(result);
}
```

Figure 4.22 Controller: SearchAndFilterController.cs

```
[HttpGet("filter-by-type/{type}")]
0 references
public ActionResult<List<SearchAndFilter>> FilterByType(string type)
{
    var result= _context.FilterByType(type);
    if( result == null)
    {
        return NotFound();
    }
    return Ok(result);
}
```

Figure 4.23 Controller: SearchAndFilterController.cs

4. User Microservice

Entity: Users.cs

```
22 references
public class Users
{
    4 references
    public string FirstName { get; set; } = null!;
    4 references
    public string LastName { get; set; }= null!;
    2 references
    public DateTime DateOfBirth { get; set; }
    3 references
    public string Password { get; set; } = string.Empty;
    [Key]
    5 references
    public string UserId { get; set; }= null!;
    4 references
    public byte[] PasswordHash { get; set; }=null!;
    4 references
    public byte[] PasswordSalt { get; set; }=null!;
    8 references
    public string Email { get; set;}=null!;
    //public string Role { get; set; } = null!; //admin or enduser

}
```

Figure 4.24 Controller: SearchAndFilterController.cs

Interface: IUserRepository.cs

```
3 references
public interface IUserRepository
{
    1 reference
    Users GetById(string id);
    3 references
    Users GetByEmail(string email);
    2 references
    Users AddUser(Users user, string password);
    1 reference
    Users Update(Users user);
    1 reference
    void Delete(Users user);
    1 reference
    bool VerifyPassword(Users user, string password);
}
```

Figure 4.25 Interface: IUserRepository.cs

Repository: UserRepository.cs

```
1 reference
public class UserRepository : IUserRepository
{
    private readonly ApplicationDbContext _context;

    0 references
    public UserRepository(ApplicationDbContext context)
    {
        _context = context;
    }

    1 reference
    public Users GetById(string id)
    {
        return _context.Users.Find(id);
    }
    3 references
    public Users GetByEmail(string email)
    {
        var result= _context.Users.FirstOrDefault(u => u.Email == email);
        if(result == null)
        {
            throw new ArgumentException("Email cannot be blank.");
        }
        return result;
    }
```

Figure 4.26 Repository: UserRepository.cs

```
public void Delete(Users user)
{
    _context.Users.Remove(user);
    _context.SaveChanges();
}

1 reference
public bool VerifyPassword(Users user, string password)
{
    if(user==null)
    {
        return false;
    }
    return VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt);

}
```

Figure 4.27 Repository: UserRepository.cs

```
private void CreatePasswordHash(string password, out byte[] passwordHash,out byte[] passwordSalt)
{
    if(password==null)
    {
        throw new ArgumentNullException("password");
    }
    if(string.IsNullOrWhiteSpace(password))
    {
        throw new ArgumentException("Value cannot be empty or whitespace only string.", nameof(password));

    }

    using (var hmac = new HMACSHA512())
    {
        passwordSalt = hmac.Key;
        passwordHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));

    }
}
```

Figure 4.27 Repository: UserRepository.cs

```
private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)
{
    if(password== null)
    {
        throw new ArgumentNullException(nameof(password));
    }
    if (string.IsNullOrWhiteSpace(password))
    {
        throw new ArgumentException("Value cannot be empty or whitespace only string.", nameof(password));

    }
    if (passwordHash.Length != 64)
    {
        throw new ArgumentException("Invalid length of password (64 byte expected).", nameof(passwordHash));
    }
    if (passwordSalt.Length != 128)
    {
        throw new ArgumentException("Invalid length of password (128 byte expected).", nameof(passwordSalt));
    }
    using (var hmac = new HMACSHA512(passwordSalt))
    {
        var computedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(password));
        for(int i=0; i<computedHash.Length; i++)
        {
            if (computedHash[i] != passwordHash[i])
            {
                return false;
            }
        }
    }
    return true;
```

Figure 4.28 Repository: UserRepository.cs

Controller: UserController.cs

```
public UserController(IUserRepository userRepository)
{
    _userRepository = userRepository;
}

[HttpPost("register")]
0 references
public IActionResult Register(Users user)
{
    var existingUser = _userRepository.GetByEmail(user.Email);
    if (existingUser != null)
    {
        return BadRequest("A user with this email already exists.");
    }

    // Generate a password hash and salt using the user's provided password
    byte[] passwordHash, passwordSalt;
    CreatePasswordHash(user.Password, out passwordHash, out passwordSalt);

    var newUser = new Users
    {
        Email = user.Email,
        FirstName = user    .FirstName,
        LastName = user.LastName,
        DateOfBirth = user.DateOfBirth,
        PasswordHash = passwordHash,
        PasswordSalt = passwordSalt
    };
```

Figure 4.29 Controller: UserController.cs

```
    var createdUser = _userRepository.AddUser(newUser,user.Password);
    return Ok(createdUser);
}

[HttpPost("login")]
0 references
public IActionResult Login(Users user)
{
    var loginuser = _userRepository.GetByEmail(user.Email);
    if (loginuser == null)
    {
        return BadRequest("Invalid email or password.");
    }

    // Verify the user's password by hashing it with the salt and comparing it to the stored hash
    var isPasswordValid = VerifyPasswordHash(user.Password, user.PasswordHash, user.PasswordSalt);
    if (!isPasswordValid)
    {
        return BadRequest("Invalid email or password.");
    }

    // Generate authentication token here and return it to the client
    var token = GenerateAuthToken(user);

    return Ok(new
    {
        user.UserId,
        user.Email,
        Token = token
    });
}
```

Figure 4.30 Controller: UserController.cs

```
private string GenerateAuthToken(Users user)
{
    // Code to generate JWT token goes here
    return "sample-token";
}

// This method generates a password hash and salt given a password
1 reference
private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)
{
    if (password == null) throw new ArgumentNullException("password");
    if (string.IsNullOrWhiteSpace(password)) throw new ArgumentException("Value cannot be empty or whitespace only string.", "password");

    using (var hmac = new System.Security.Cryptography.HMACSHA512())
    {
        passwordSalt = hmac.Key;
        passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
    }
}

// This method verifies a password hash given a password and salt
1 reference
private bool VerifyPasswordHash(string password, byte[] storedHash, byte[] storedSalt)
{
    if (password == null) throw new ArgumentNullException("password");
    if (string.IsNullOrWhiteSpace(password)) throw new ArgumentException("Value cannot be empty or whitespace only string.", "password");
    if (storedHash.Length != 64) throw new ArgumentException("Invalid length of password hash (64 bytes expected).", "passwordHash");
    if (storedSalt.Length != 128) throw new ArgumentException("Invalid length of password salt (128 bytes expected).", "passwordSalt");

    using (var hmac = new System.Security.Cryptography.HMACSHA512(storedSalt))
    {
        var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
        for (int i = 0; i < computedHash.Length; i++)
        {
            if (computedHash[i] != storedHash[i]) return false;
        }
    }

    return true;
}
```

Figure 4.31 Controller: UserController.cs

5. Application DbContext

```
public class ApplicationDbContext : DbContext
{
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>options):base(options)
    {

    }
    6 references
    public DbSet<ClothingItem> ClothingItems { get; set; } = null!;
    4 references
    public DbSet<SearchAndFilter> SearchandFilter { get; set; } = null!;
    6 references
    public DbSet<Outfit> Outfits { get; set; } = null!;
    5 references
    public DbSet<Users> Users { get; set; } = null!;
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<ClothingItem>()
                    .HasKey(c => c.Id);
    }

}
```

Figure 4.32 Application DbContext

6. Adding and Updating Migrations

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "SearchandFilter",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Type = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Colour = table.Column<string>(type: "nvarchar(max)", nullable: false),
            Size = table.Column<string>(type: "nvarchar(max)", nullable: false),
            OutfitName = table.Column<string>(type: "nvarchar(max)", nullable: false),
            OutfitIfd = table.Column<int>(type: "int", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_SearchandFilter", x => x.Id);
        });
```

Figure 4.33 Adding and updating Migrations

```
migrationBuilder.CreateTable(
    name: "Users",
    columns: table => new
    {
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        FirstName = table.Column<string>(type: "nvarchar(max)", nullable: false),
        LastName = table.Column<string>(type: "nvarchar(max)", nullable: false),
        DateOfBirth = table.Column<DateTime>(type: "datetime2", nullable: false),
        Password = table.Column<string>(type: "nvarchar(max)", nullable: false),
        PasswordHash = table.Column<byte[]>(type: "varbinary(max)", nullable: false),
        PasswordSalt = table.Column<byte[]>(type: "varbinary(max)", nullable: false),
        Email = table.Column<string>(type: "nvarchar(max)", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Users", x => x.UserId);
    });

migrationBuilder.CreateTable(
    name: "Outfits",
    columns: table => new
    {
        OutfitId = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        OutfitName = table.Column<string>(type: "nvarchar(max)", nullable: false),
        OutfitImageUrl = table.Column<string>(type: "nvarchar(max)", nullable: false),
        CreatedAt = table.Column<DateTime>(type: "datetime2", nullable: false),
        UpdatedAt = table.Column<DateTime>(type: "datetime2", nullable: false),
        UsersUserId = table.Column<string>(type: "nvarchar(450)", nullable: false)
```

Figure 4.34 Adding and updating Migrations

```
    constraints: table =>
    {
        table.PrimaryKey("PK_Outfits", x => x.OutfitId);
        table.ForeignKey(
            name: "FK_Outfits_Users_UsersUserId",
            column: x => x.UsersUserId,
            principalTable: "Users",
            principalColumn: "UserId",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateTable(
    name: "ClothingItems",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Type = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Colour = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Size = table.Column<string>(type: "nvarchar(max)", nullable: false),
        ImageUrl = table.Column<string>(type: "nvarchar(max)", nullable: false),
        OutfitId = table.Column<int>(type: "int", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ClothingItems", x => x.Id);
        table.ForeignKey(
            name: "FK_ClothingItems_Outfits_OutfitId",
            column: x => x.OutfitId,
            principalTable: "Outfits",
            principalColumn: "OutfitId");
    });
```

Figure 4.35 Adding and updating Migrations

```
        migrationBuilder.CreateIndex(
            name: "IX_ClothingItems_OutfitId",
            table: "ClothingItems",
            column: "OutfitId");

        migrationBuilder.CreateIndex(
            name: "IX_Outfits_UsersUserId",
            table: "Outfits",
            column: "UsersUserId");
    }

    /// <inheritdoc />
    0 references
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "ClothingItems");

        migrationBuilder.DropTable(
            name: "SearchandFilter");

        migrationBuilder.DropTable(
            name: "Outfits");

        migrationBuilder.DropTable(
            name: "Users");
    }
```

Figure 4.36 Adding and updating Migrations

7. ApplicationDbContextSnapshot.cs

```
0 references
protected override void BuildModel(ModelBuilder modelBuilder)
{
#pragma warning disable 612, 618
    modelBuilder
        .HasAnnotation("ProductVersion", "7.0.5")
        .HasAnnotation("Relational:MaxIdentifierLength", 128);

    SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

    modelBuilder.Entity("DataAccessLayer.Models.ClothingItem", b =>
        {
            b.Property<int>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("int");

            SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

            b.Property<string>("Colour")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("ImageUrl")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("Name")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<int?>("OutfitId")
                .HasColumnType("int");

            b.Property<string>("Size")
                .IsRequired()
                .HasColumnType("nvarchar(max)");
```

Figure 4.37 ApplicationDbContextSnapshot.cs

```
        b.HasKey("Id");

        b.HasIndex("OutfitId");

        b.ToTable("ClothingItems");
    });

    modelBuilder.Entity("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Authentication___Authorization.Users", b =>
    {
        b.Property<string>("UserId")
            .HasColumnType("nvarchar(450)");

        b.Property<DateTime>("DateOfBirth")
            .HasColumnType("datetime2");

        b.Property<string>("Email")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("FirstName")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("LastName")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Password")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<byte[]>("PasswordHash")
            .IsRequired()
            .HasColumnType("varbinary(max)");
```

Figure 4.38 ApplicationDbContextSnapshot.cs

```
            b.Property<byte[]>("PasswordHash")
                .IsRequired()
                .HasColumnType("varbinary(max)");

            b.Property<byte[]>("PasswordSalt")
                .IsRequired()
                .HasColumnType("varbinary(max)");

            b.HasKey("UserId");

            b.ToTable("Users");
        });

    modelBuilder.Entity("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Outfit", b =>
        {
            b.Property<int>("OutfitId")
                .ValueGeneratedOnAdd()
                .HasColumnType("int");

            SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("OutfitId"));

            b.Property<DateTime>("CreatedAt")
                .HasColumnType("datetime2");

            b.Property<string>("OutfitImageUrl")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("OutfitName")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<DateTime>("UpdatedAt")
                .HasColumnType("datetime2");
```

Figure 4.39 ApplicationDbCvontextSnapshot.cs

```
        b.Property<string>("UsersUserId")
            .IsRequired()
            .HasColumnType("nvarchar(450)");

        b.HasKey("OutfitId");

        b.HasIndex("UsersUserId");

        b.ToTable("Outfits");
    });

modelBuilder.Entity("Virtual_Wardrobe_Management_System.Data_Layer.Entities.SearchAndFilter", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("int");

        SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

        b.Property<string>("Colour")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Name")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<int>("OutfitIfd")
            .HasColumnType("int");

        b.Property<string>("OutfitName")
            .IsRequired()
            .HasColumnType("nvarchar(max)");
```

Figure 4.40 ApplicationDbContextSnnapshot.cs

```
            b.Property<string>("Size")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.Property<string>("Type")
                .IsRequired()
                .HasColumnType("nvarchar(max)");

            b.HasKey("Id");

            b.ToTable("SearchandFilter");
        });

    modelBuilder.Entity("DataAccessLayer.Models.ClothingItem", b =>
        {
            b.HasOne("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Outfit", null)
                .WithMany("Items")
                .HasForeignKey("OutfitId");
        });

    modelBuilder.Entity("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Outfit", b =>
        {
            b.HasOne("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Authentication___Authorization.Users", "Users")
                .WithMany()
                .HasForeignKey("UsersUserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();

            b.Navigation("Users");
        });

    modelBuilder.Entity("Virtual_Wardrobe_Management_System.Data_Layer.Entities.Outfit", b =>
        {
            b.Navigation("Items");
        });
```

Figure 4.41 ApplicationDbContextSnnapshot.cs

8. Establishing Connection strings

```
a: https://json.schemastore.org/appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=LTIN140969\\SQLEXPRESS ;Database=VirtualWardrobe ;TrustServerCertificate=True; User ID=admin;Password=Cognizant@1234"
  }
}
```

Figure 4.42 Establishing Connection Strngs

9. Program.cs File

```
0 references
internal class Program
{
    0 references
    private static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.

        builder.Services.AddControllers();
        builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseHttpsRedirection();

        app.UseAuthorization();

        app.MapControllers();

        app.Run();
    }
}
```

Figure 4.43 Program.cs File

10. After Build and Run

Opening Swagger

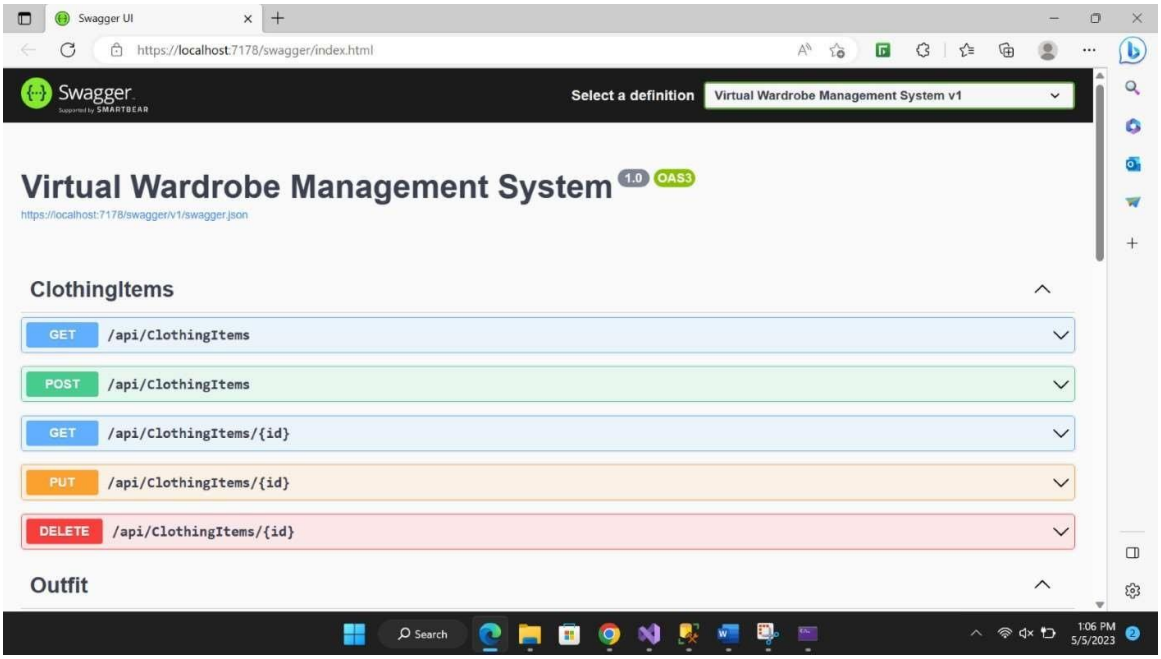

Figure 4.44 Opening Swagger

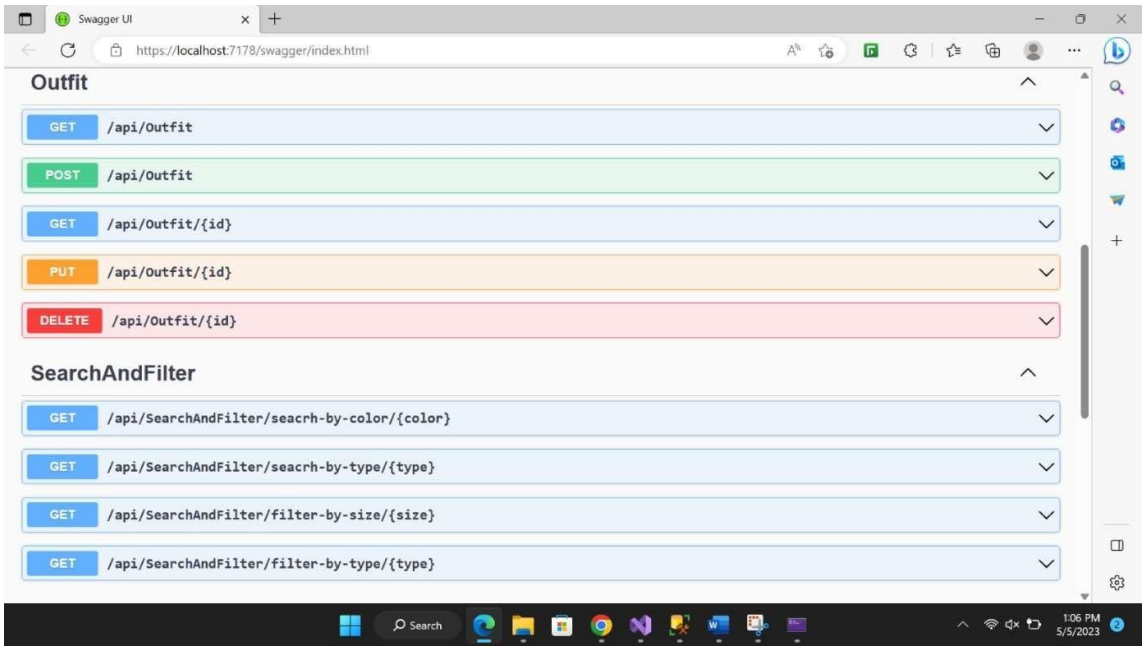

Figure 4.45 Opening Swagger

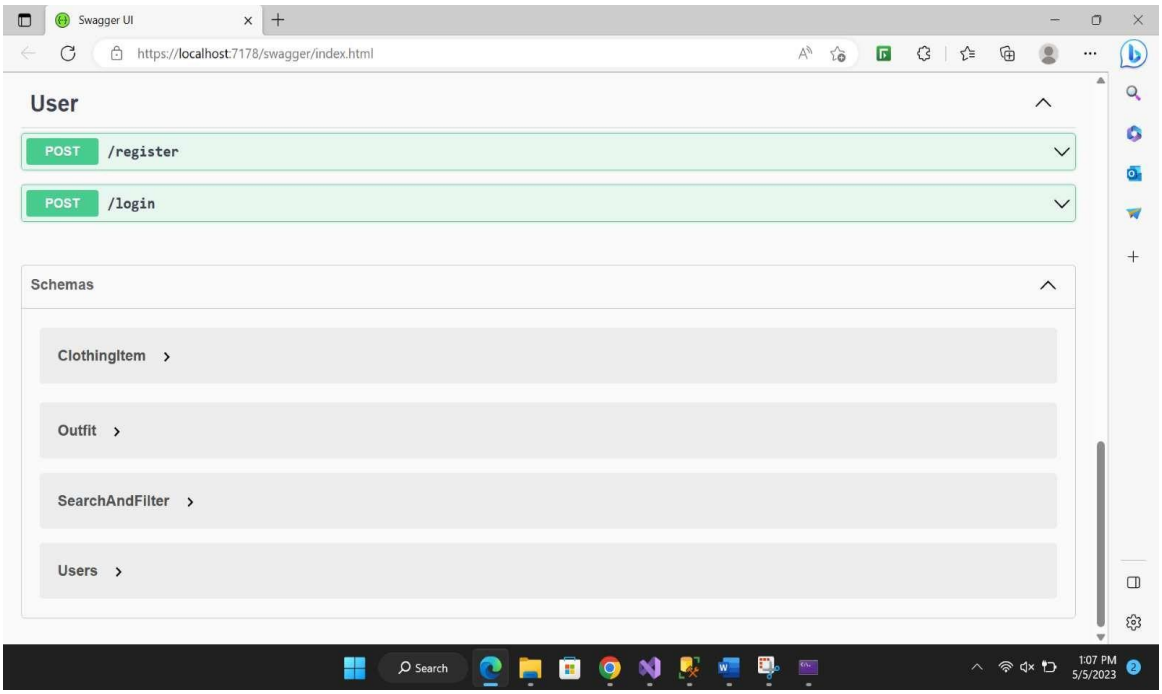

Figure 4.46 Opening Swagger

11. Database Creation

SQL Server Management Studio

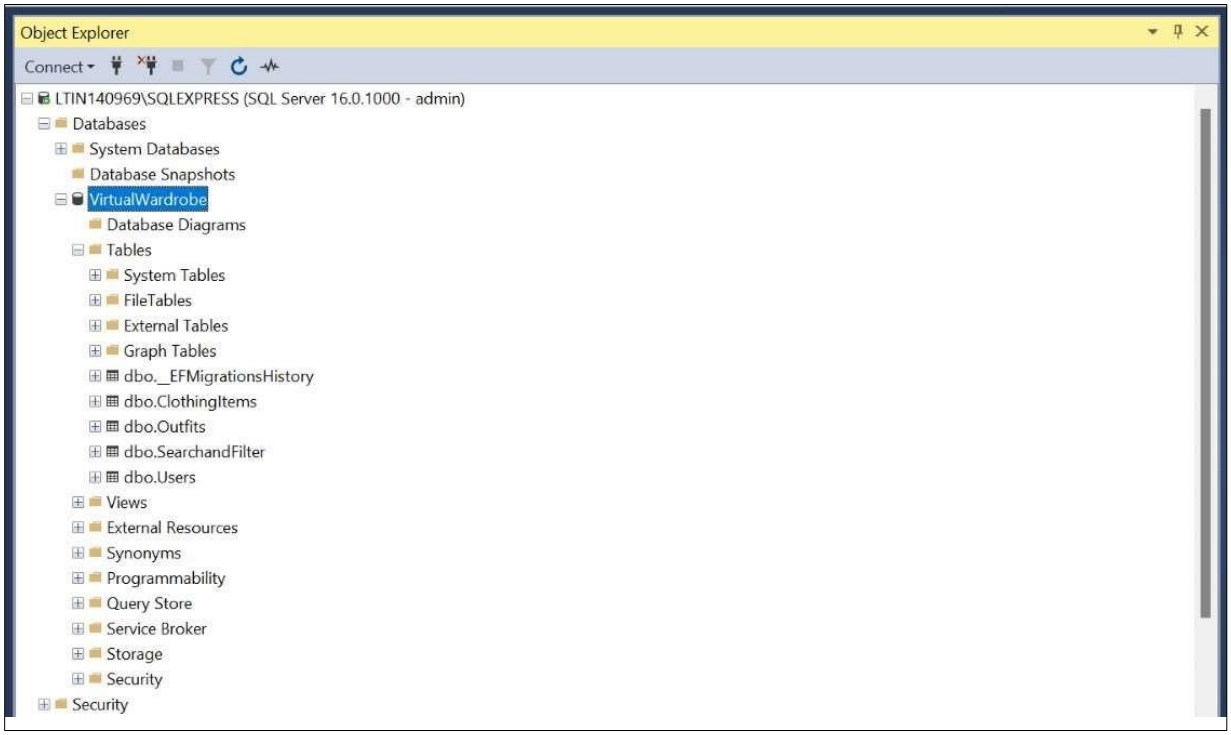

Figure 4.47 SQL Server Management Studio

CHAPTER-5 CONCLUSIONS

5.1 Conclusions

The implementation of a microservices-based architecture for an online clothes shop demonstrated the benefits of a modular and scalable approach to software design. Each microservice is responsible for a particular job and may be established, deployed, and maintained independently from the others.

The usage of the EF as an Object-Relational Mapping (ORM) tool has greatly simplified database management, allowing for rapid development and prototyping. Because of the use of data annotations and migrations, the database schema was straightforward to build, maintain, and upgrade. Microservices for search and filtering, user identity and permission, and outfit creation have all been deployed, and they have all resulted in critical features that are required for every e-commerce siteThese microservices may be readily tailored to the specific needs of the organisation. The implementation of RESTful APIs has enabled the microservices to be exposed to clients thorugh standadised and widely used interface.

Overall, the project highlighted the beneffits of services architecture, EF, REST APIs, and testing for the construction of a sclable and fexible e-comerce platforms.

5.2 Future Scope

Based on the project's findings, below are some possible future directions or ideas for improvement:

- The introduction of a machine learning-powered recommendation engine to provide clients with personalised apparel suggestions based on their interests, previous purchases, and platform usage.
- Including more particular product information, such as fabric composition, care requirements, and country of origin, to help users make more informed purchase decisions.
- The inclusion of a chatbot to respond to users' inquiries and concerns in real time. This has the potential to improve customer happiness and overall user experience.

User data such as purchase history, browsing patterns, and comments would generally need to be acquired and reviewed in order to construct a machine learning model for a recommendation system. The model may then offer personalised suggestions for each user based on their data. Various strategies, like collaborative filterig, content-based filtering, or a mix of the two, might be

used to train the model. The suggestions might be published on the user's home page or on a sepaate recommendaions page, and the usr may offer input on the recommendatons to help the model become more refined. The recommendaton system might be regularly updatd and refined over time to give users with more accurte and relevant suggestions.

REFERENCES

[1] H. Zhang, X. Li, J. Li, and H. Li, "A virtual wardrobe system based on image recognition and recommendation," in Proceedings of the Journal of Fashion Technology & Textile Engineering, vol. 6, no. 1, pp. 1-7, 2018.

[2] X. Li, J. Li, H. Zhang, and H. Li, "A mobile application for virtual mixing and matching of clothes," in Proceedings of the International Journal of Human-Computer Studies, vol. 123, pp. 1-10, 2019.

[3] Y. Chen, Q. Guo, X. Liu, and Y. Wang, "Deep learning-based clothing item recognition and categorization for virtual wardrobe systems," in Proceedings of IEEE Access, vol. 8, pp. 186275-186285, 2020.

[4] X. Xie, J. Wang, Y. Li, and M. Chen, "A recommendation algorithm for outfit combinations in virtual wardrobe management systems," in Proceedings of ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 17, no. 3, pp. 1-21, 2021.

[5] J. Liu, Y. Zhang, L. Cao, and Y. Ma, "A virtual dressing room system based on augmented reality and 3D garment modeling," in Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), pp. 1-6, 2020.

[6] C. Wang, W. Huang, Y. Li, and M. Zhang, "Mobile application for virtual wardrobe management with cloud-based storage and synchronization," in Proceedings of the International Journal of Information Management, vol. 54, pp. 1-10, 2020.

[7] X. Tang, Z. Liu, J. Chen, and B. Zhou, "Gesture-based interaction system for virtual wardrobe management," in Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 3, no. 2, pp. 1-23, 2019.

[8] S. Kim, J. Park, and S. Lee, "User acceptance and satisfaction of a virtual wardrobe management system," in Proceedings of Computers in Human Behavior, vol. 127, pp. 1-12, 2022.