Virtual Outfit Manager

Final Report

Sanket Karna and Roman Gofman

CSS 497

Professor A. Retik

August 18th, 2023

Table of Contents

[**Introduction 3**](#_i4nxw5o46ypy)

[Background 3](#_61czzhgr3zo9)

[Goals 3](#_hhla21sdsqhl)

[**Main Project 4**](#_ufgxi57ni8vu)

[Requirements 4](#_t3czkzjb1sju)

[Design 4](#_ohj9kj8pq6th)

[Architecture Style 4](#_bodf3ft1fodg)

[Database Design 5](#_28fjbl3z9410)

[Flask Framework 6](#_9wxopb1n9c15)

[Cloud implementation 7](#_v0s593xhds2k)

[Development 7](#_nb0t4rolsim6)

[Database Development 7](#_hnkdcpg3o5u)

[Backend 8](#_p6nivmv17686)

[ODBC Database Connection 8](#_mqou1hrvjdjs)

[Account Creation and Login 8](#_nte5z8n3n4ds)

[Wardrobe Management 9](#_l78vpt4ktlwp)

[Image Upload 9](#_ojmmfhgayqj6)

[Frontend 9](#_vafn98hl5nn9)

[Account Creation and Login 9](#_6ay9fvcrxepv)

[Image Upload Form 11](#_lufvmwvoo1bj)

[Displaying Query Results 11](#_wt39g1na4bvo)

[Wardrobe Management 12](#_l840phum24v9)

[Outfit Management 13](#_mwmmlhmrny5i)

[Outfit Creation 13](#_8vyrt1be2uvh)

[Results 14](#_bm4vwylj3dud)

[Testing 14](#_38sar6et01rs)

[Development testing 14](#_6h9i83pv4pko)

[User testing feedback 14](#_rc2njh8agl6n)

[Achieving 15](#_hh08h57o26fg)

[Skills Learned 15](#_kzo0i4xvajzj)

[Accomplishments 15](#_e9qcqpxa9ckm)

[**Conclusion 16**](#_1jl36v07nx)

[Final Thoughts 16](#_gwr1h36dnsr6)

[Next Steps 16](#_r9n05wvfzgdp)

# Introduction

## Background

In a world saturated with clothing options, the process of curating the perfect outfit has become an unexpectedly daunting task. This phenomenon arises from the overwhelming abundance of choices, which leaves individuals grappling with decision fatigue and choice paralysis. This challenge extends beyond the realm of fashion and taps into the broader psychology of choice—the more options we have, the more challenging it becomes to make decisions. This struggle to visualize outfits resonates deeply with modern life's demand for efficiency and time management, which sparked the idea for this project in CSS 490 (Software Entrepreneurship). This idea aims to simplify the process of outfit selection by providing a platform for users to create and manage outfits, not only transforming wardrobes but also addressing a fundamental challenge of contemporary society.

## Goals

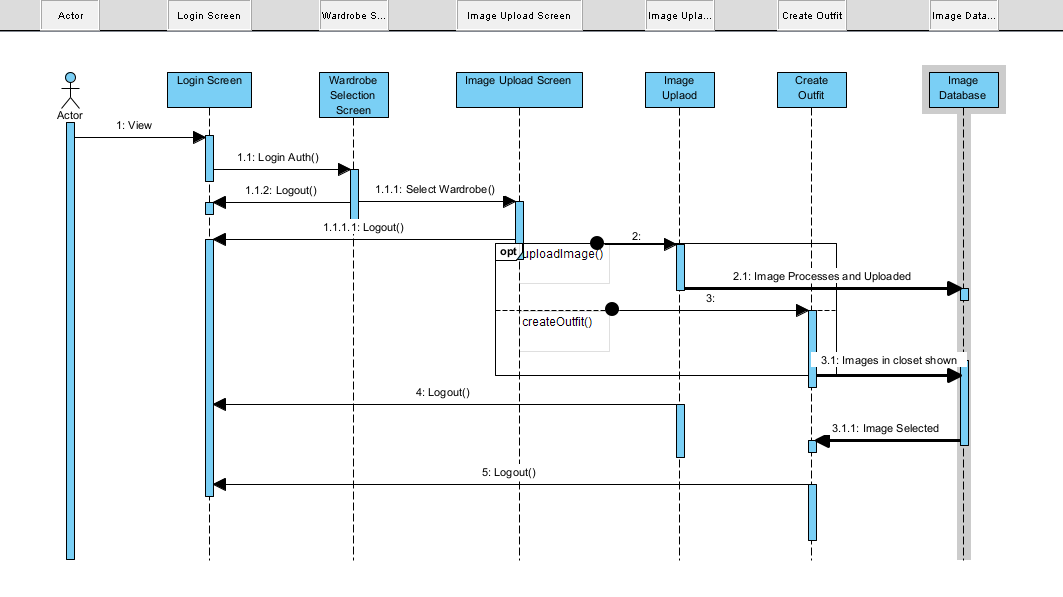
The goal of this project was to create a user-friendly outfit design software by focusing on efficient clothing management and outfit creation from virtual wardrobes. Our approach involves user-centric research, a three-tier architecture, and the utilization of Google's Vision API for image processing and categorization. The technology stack is comprised of Flask, HTML/CSS/JavaScript, Python, utilizing object-oriented principles, and Azure SQL Databases for a streamlined unilateral functionality. Collaboration was facilitated through GitHub, and regular meetings were held to address challenges and share progress. Key milestones include user account management, image processing, serialization, outfit creation, and robust UI-to-database connections. Our primary goal is to enhance user experience by simplifying outfit planning and delivering an engaging interface.

In summary, our project aims to develop user-friendly outfit design software through careful research and modern technology. We'll maintain collaboration and track milestones while focusing on enhancing user experience, making outfit planning and management intuitive and enjoyable.

# Main Project

## Requirements

Our project entails the development of a comprehensive clothing management system with a range of programming deliverables. It will include essential features such as user login functionality and user creation, enabling individuals to establish personalized accounts. Users will be able to create multiple wardrobes tailored to different occasions or preferences, select and manage items within these wardrobes, and seamlessly delete unwanted wardrobes. Additionally, the system will support image upload capabilities, employing sophisticated image analysis techniques to generate accurate descriptions and streamline color tones. An innovative outfit visualizer will empower users to effortlessly combine and visualize outfits, while enabling outfit creation from scratch. Unit testing will be rigorously implemented to ensure smooth user interactions, and functionality for saving, loading, and deleting user-generated outfits will be integrated for a seamless experience. Refer *Fig 1.* For example, use case.  
  
*Figure 1: Sequence Diagram*



## 

## Design

### Architecture Style

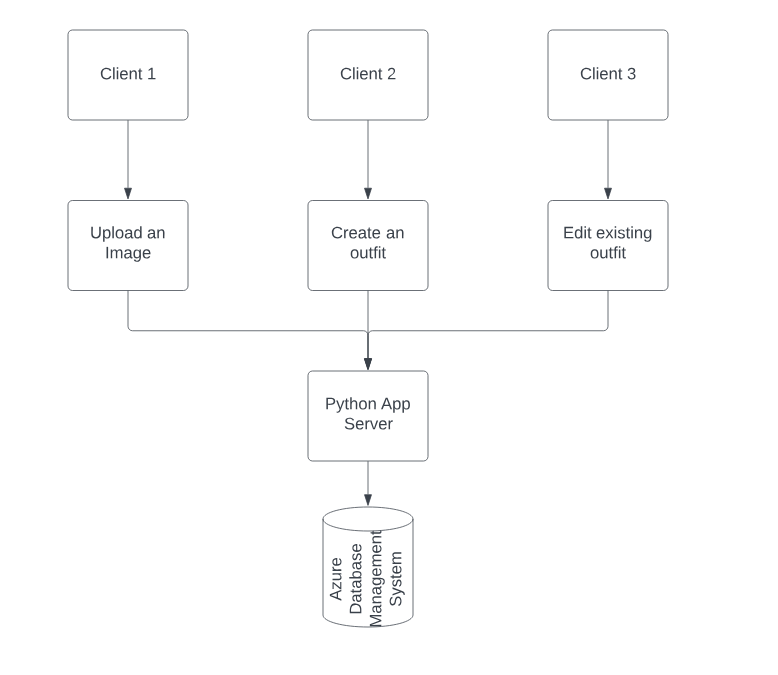
Tier 1: Presentation Layer, the presentation layer is responsible for the user interface and user interactions. It's where users interact directly with the software.

Tier 2: Application Logic Layer, the application logic layer handles the processing of data and logic behind user requests. It acts as an intermediary between the presentation layer and the data storage layer.

Tier 3: Data Storage Layer, the data storage layer is responsible for storing and managing data used by the application.

In this 3-tier architecture, the layers work together to create a well-organized and efficient flow of data and interactions. The presentation layer interacts with users, the application logic layer processes and manages data and requests, and the data storage layer securely stores the relevant information. This separation of concerns enhances modularity, scalability, and maintainability of your outfit design software. (Reference *Fig*. 2)

*Figure 2: 3-Tier Architecture*

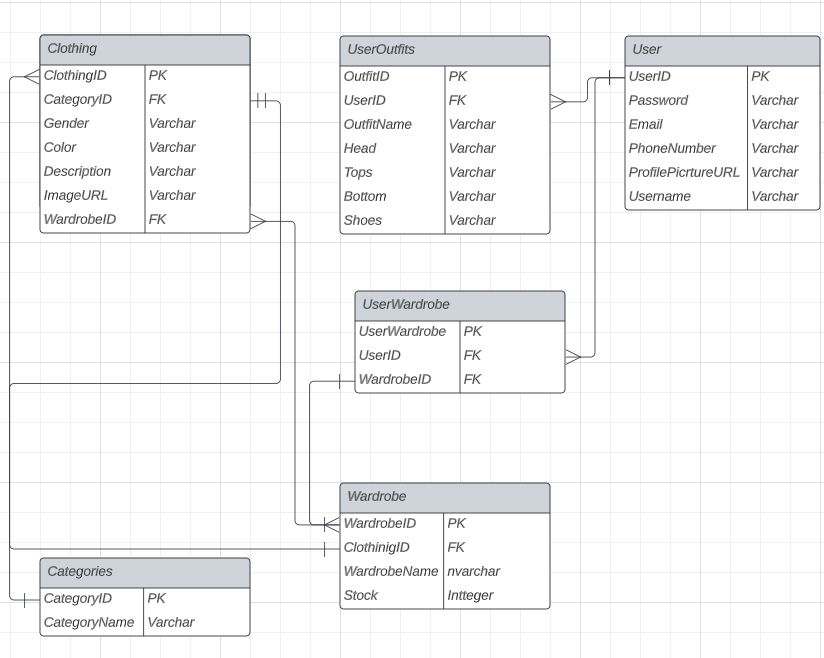


### Database Design

The database design is a SQL relational database built in MySQL and deployed to Azure SQL databases. The schema was based around a layer of abstraction for the user’s data. Specifically in terms of how the relational database was built, the foreign keys that were most common were the USERID and the WARDROBEID these were used to make sure that every user saw only their wardrobes as a result only the clothes and outfits connected to their USERID or other associated foreign keys (Reference *Fig.* 3).

We ended up ensuring that the Database was normalized to the 3rd normal form meaning that as much data was isolated in its own tables as possible, but it was not fully normalized for the sake of making the database easier to access and make resulting queries simpler overall. The main feature of the database is the self-populating key values ensuring a unique key each and every time that was handled by the database rather than our internal code.

*Figure 3: Entity Relationship Diagram*



### Flask Framework

The flask framework provided a base for us to start building our project. It provides much assistance with the often-difficult parts of working with webapps. Specifically routing data from one page to another or between respective files. This was a significant difficulty initially wanting to connect a front-end user interface built in HTML and CSS, to a Python back end through JavaScript. Flask provides a way to do this using route in which data can be passed and a different view can be rendered. Especially when it came to introducing data coming from several different sources, namely the User and the Cloud.

Finally, what was likely the single most valuable function of the framework was the integrated ability to work in a python virtual environment allowing for local development and debugging before deploying to the cloud. This was also made easier by flask as the Azure CLI had an existing system in place for deploying a web app which ignoring any extraneous circumstances worked without a hitch.

### Cloud implementation

Our app's cloud implementation thrives within the Azure ecosystem, comprising the web app, Azure blob storage, and the database. Hosted on Azure, the web app offers global accessibility, while Azure blob storage securely houses images, ensuring their availability throughout the application.

This implementation excels due to Azure's powerful infrastructure, providing a robust foundation for the app's growth. The deployment grants universal access to the website, embracing users worldwide. To bolster security, the database restricts access to specific IPs, safeguarding sensitive data while fostering user trust as well as access through other Azure Services. The cloud's appeal lies in its scalability, a virtue amplified by Azure. Our app can effortlessly accommodate increased demand, laying the groundwork for expansion without compromising performance.

## Development

### Database Development

We seamlessly integrated the creation of our Azure SQL database with the powerful tools of Azure Data Studio. This involved meticulously designing and configuring our database's structure through SQL scripting, encompassing essential features like user authentication, wardrobe creation, outfit selection, image upload, and dynamic outfit visualization. Leveraging Azure Data Studio, I found myself crafting and executing SQL queries to seamlessly manage our database, ensuring a user-friendly experience, and streamlined data manipulation. Moreover, Azure Data Studio became our go-to for version control, fine-tuning performance, and deploying updates – offering us a unified, efficient environment to shape our app into a comprehensive and impactful outfit management solution.

### Backend

#### ODBC Database Connection

PyODBC provides a straightforward and versatile way to establish connections between Python applications and Azure SQL databases. Leveraging this library, developers can effortlessly create connections by specifying the server address, database name, and authentication credentials. PyODBC seamlessly bridges the gap between Python and SQL, allowing the execution of SQL queries and data manipulation operations directly from Python code. Its adaptability enables compatibility with various SQL database systems, including Azure SQL, empowering developers to retrieve, update, or insert data with ease. This connection mechanism offered by PyODBC forms the foundation for building dynamic and data-driven applications that interact seamlessly with Azure SQL databases.

#### Account Creation and Login

In our code, we've implemented the signup and login authentication functionalities within a Flask web application. Within the /signup route, I retrieve user input data such as first name, last name, email, and password from a signup form. By using PyODBC, I establish a connection to our Azure SQL database and insert the collected data into the USER table. Subsequently, I close the connection after the data is committed. Similarly, in the /loginauth route, I extract user login credentials and create a database connection. Through a validation query, I verify if the provided username and password exist in the database. Upon successful authentication, I store the user's current ID and retrieve existing wardrobe data. For unsuccessful login attempts, I display an error message. The PyODBC library helps facilitate database interactions in both routes, aiding me in connecting to the database, executing SQL queries, and rendering template responses. Our code structure ensures an organized approach to managing user authentication and Azure SQL database interactions while effectively integrating these processes into our Flask application.

#### Wardrobe Management

#### Image Upload

Our approach to handling image uploads and processing in our web app involved several key steps. When a user uploads an image using the designated button, the chosen file is securely stored in Azure blob storage, ensuring data integrity. The central feature of our application was the seamless integration of Google's Vision API, which enables advanced image processing tasks like object and label detection.

This integration with the Vision API formed the backbone of our processing pipeline. Images were systematically passed through the API, allowing us to extract essential data from them. Notably, detected objects underwent a categorization process using a cleverly implemented Jaccard index algorithm. This algorithm efficiently assigned uploaded items to categories such as Headwear, Tops, Bottoms, or Shoes. Additionally, detected labels played a crucial role in forming detailed image descriptions to enhance the user experience.

We also harnessed external resources to enrich our software's capabilities. By incorporating APIs like the Color API, we could retrieve accurate color information. However, to ensure user-friendly color names, we employed a customized nearest neighbor search algorithm. Finally, the culmination of these processes led to the seamless integration of meticulously structured data into a database. Attributes such as CategoryID, Gender, Color, Description, ImageURL, and WardrobeID were stored, ensuring smooth synchronization with the user's wardrobe.

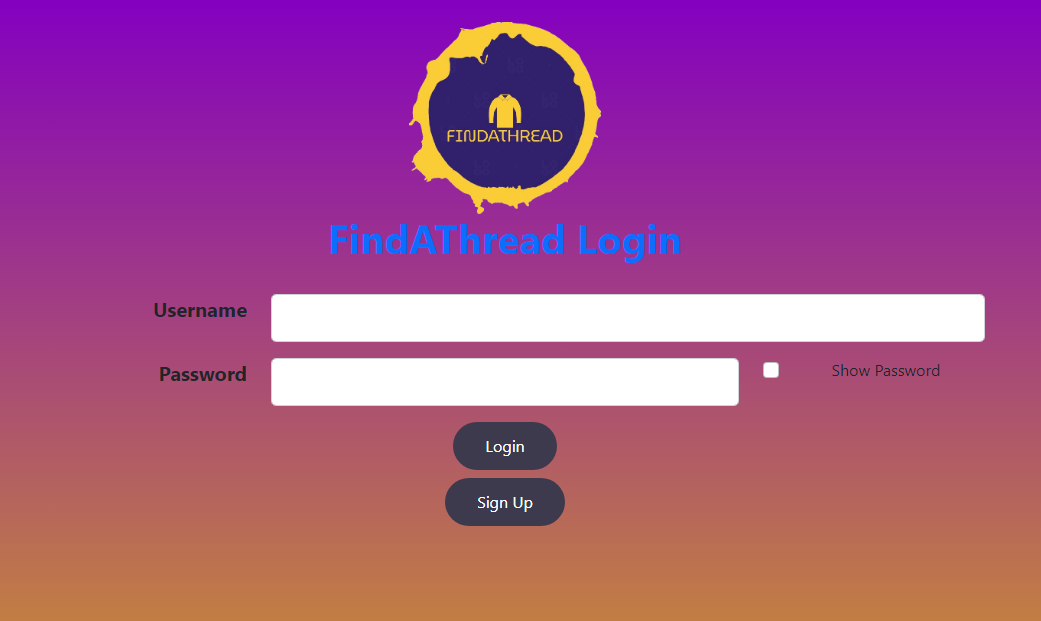
### Frontend

#### Account Creation and Login

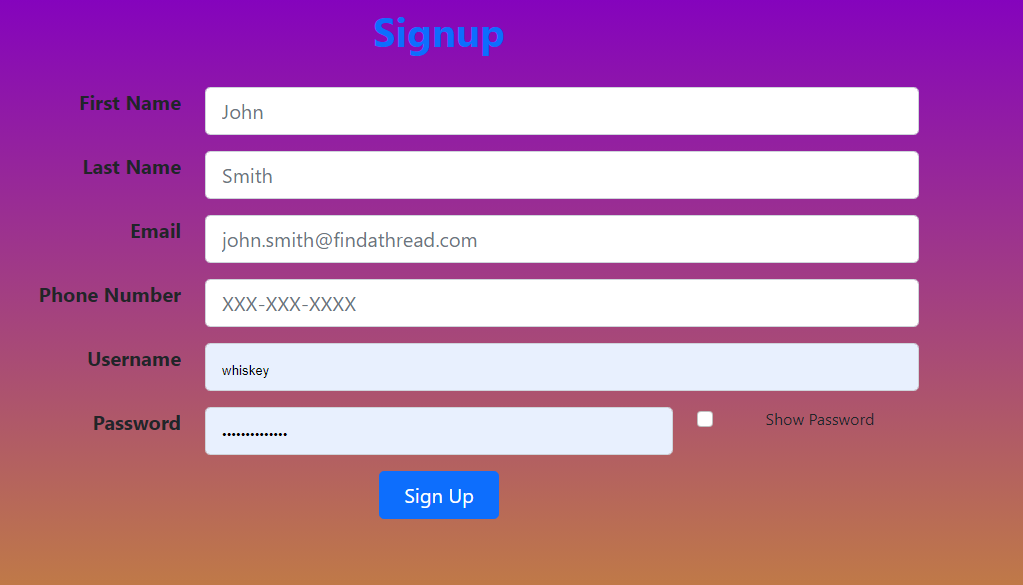
Our platform offers efficient account creation, enabling users to register quickly by providing the necessary information: name, email, username, password, and telephone number. Once the account is successfully created, users are immediately redirected to the login page, with their newly installed credentials.

The login process further exemplifies our commitment to user friendliness and safety. When users enter their login details, authenticated credentials direct them to their personal details page with ease, ensuring easy access to their personal details. In case they provide incorrect information, our platform immediately provides users with intuitive information, and guides individuals to a successful login experience. This approach not only ensures that users are notified quickly but also points them in the right direction, allowing them to interact positively with our platform. (Refer *Fig.* 4, *Fig.*5.)

*Figure 4: Login Page*



*Figure 5: Signup Page*



#### Image Upload Form

Our webpage streamlines image uploads with user convenience in mind. Through an intuitive box, users can click or drag and drop images, limited to JPG, JPEG, PNG, and JFIF file types. After uploading, users smoothly progress to the next step, where they indicate the clothing item's gender. Once submitted, the image is seamlessly uploaded. Upon success, a JavaScript-driven flash message, powered by JSONify, adds a touch of satisfaction. This user-centric approach ensures an easy to use and understand image upload experience.

#### Displaying Query Results

Displaying query results in our HTML involves seamlessly combining backend data with frontend design. Through a templating engine, we effortlessly integrate fetched information into our webpage's structure. In essence, our approach seamlessly merges backend data with frontend templates, creating an intuitive and interactive display of query results right within our HTML pages. (Refer *Fig 6.*)

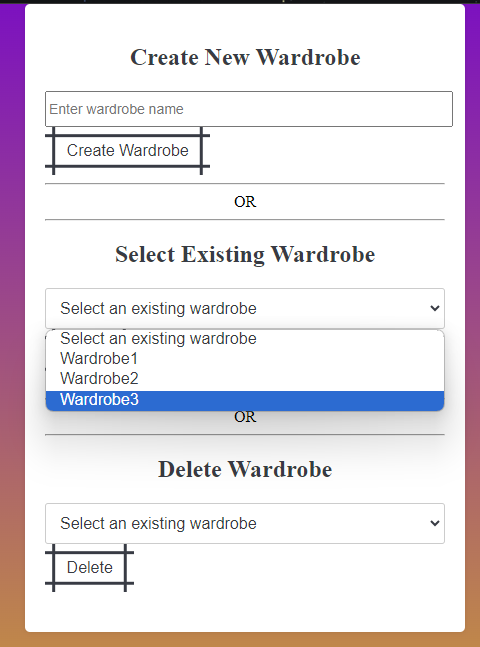
*Figure 6: Tops Page*



##### Wardrobe Management

In our web application's wardrobe management functionality, I've designed an intuitive user interface that facilitates creating, selecting, and deleting wardrobes. Users can effortlessly create new wardrobes by entering a name and clicking the "Create Wardrobe" button. When selecting an existing wardrobe, a dropdown menu displays available options generated through backend templating. To ensure accuracy, JavaScript validation prompts an alert if no wardrobe is chosen. Similarly, for wardrobe deletion, users select a wardrobe from the dropdown, with validation safeguards in place to prevent accidental deletions. The integration of JavaScript alerts in each form guarantees real-time validation and guides users through the wardrobe management process, contributing to a seamless and user-friendly experience.(Reference *Fig. 7*)

*Figure 7: Wardrobe Management Page*



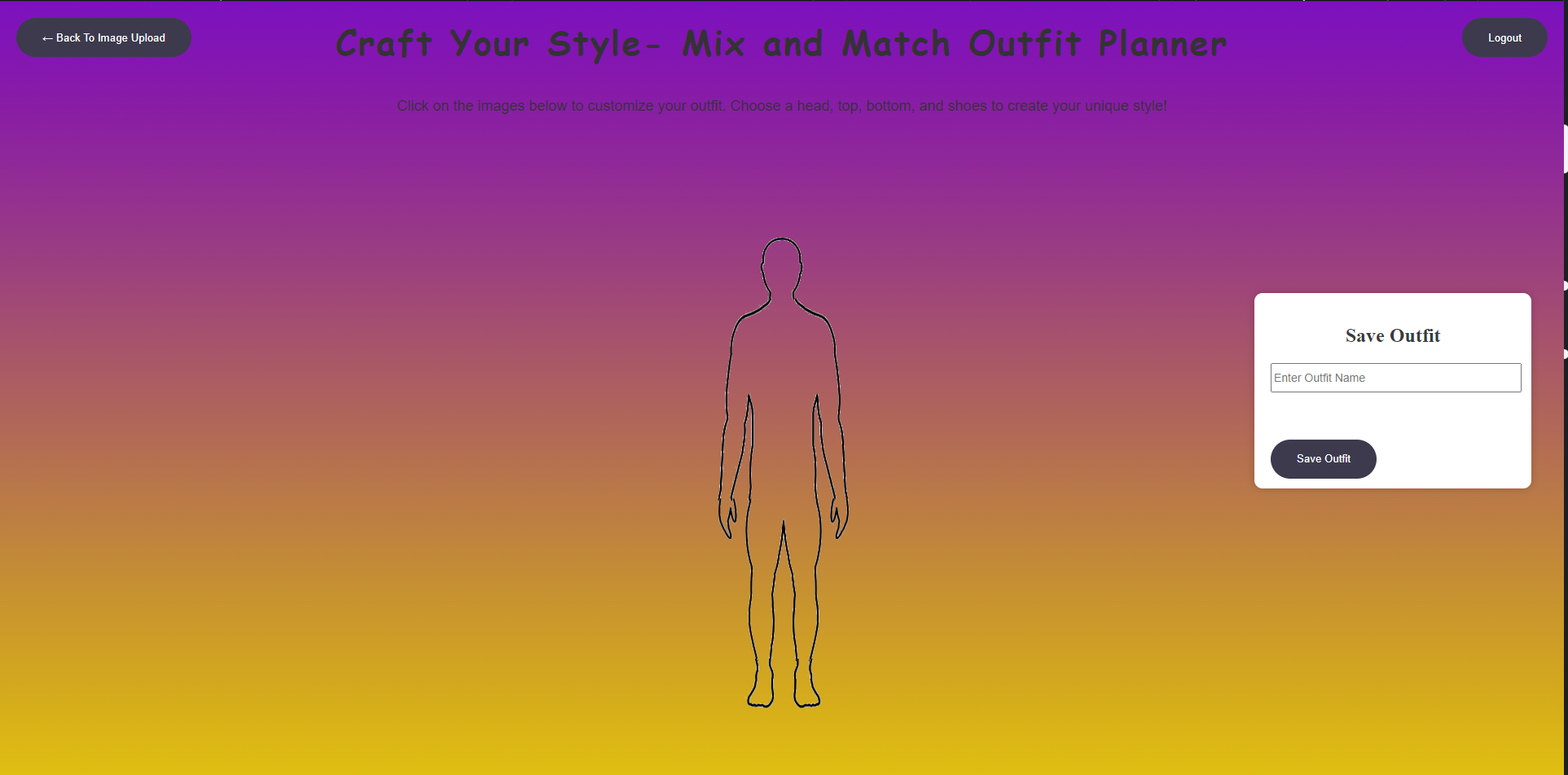
##### Outfit Management

Outfit management works much the same to how the above wardrobe management system works for instance the main goal is to tailor it to a user's current action to delete, create, and load are all actions on the main page before outfit creation as the user would make the decision before going through and updating an outfit. Then there is the save outfit feature which gives the user the freedom to select their own unique name for their outfit, so it is more convenient to find when they need it. The save option allows a user to have a set of outfits for any event they wish without having to recreate the given outfit. For this reason, outfits are connected through USERID as a foreign key rather than to the wardrobe because the user wants the outfits readily available.

##### Outfit Creation

This portion is 100% up to the user to customize; they choose whichever items they want in their ideal outfit and can save, load, and delete these outfits. The user chooses them by clicking on one of the four main images which will pull up all images that are both in the current wardrobe as well as in the current selected category. The biggest difficulty with this was the alignment of the images because the CSS was not aligning the image containers; this then plagued us with the UI because we initially had buttons which we changed to be the images and as a result had alignment issues as a whole. In the end we ended up finding a fix for this in making separate containers for the images and the buttons respectively.(reference *Fig. 8*)

*Figure 8: Outfit Creation Page*



##### Results

### Testing

#### Development testing

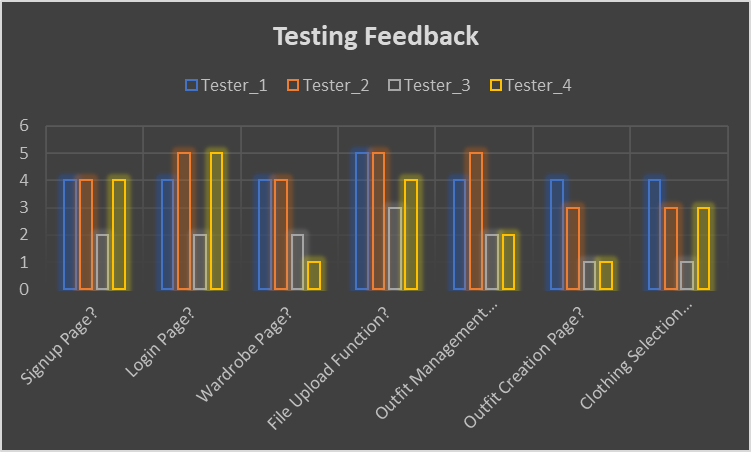
Testing has been central to the overall development of our application, ensuring reliability and performance. We adopted a comprehensive testing approach, including unit testing, query testing and integration testing, each tailored to the specific functionality of the application.

Unit testing has thoroughly examined individual components, such as user accounts and wardrobe management. Query testing focused on seamless interaction with the database, while integration testing ensured that features such as Vision API image processing, clothing rendering etc. worked properly. This careful testing process provides users a seamless experience as they engage in account creation, photo uploading and outfit design. By prioritizing thorough testing, we believe in delivering reliable and error-free applications to users.

#### User testing feedback

The feedback provided by the various users was very much in line with our larger goals. While the performance of our application was commendable within the time and resources available, it is acknowledged that additional time and resources could have allowed us to further enhance its capabilities. We thoughtfully added user interfaces, enhanced the app's user experience by improving the interactive buttons, and ensured consistency across pages. In response to user feedback, we have also enhanced the clothing settings page with clear instructions to ensure users navigate the process smoothly. This common approach to customizing the application based on user insights underlines our commitment to providing the best possible user experience. (Refer *Fig. 9: Testing Feedback)*

*Figure 8: Testing Feedback*



### Achieving

#### Skills Learned

We gained various skills in this project which greatly improved our capabilities. Involved in frontend development, we know HTML, CSS and JavaScript to create dynamic user interfaces with a deep understanding of design principles and frameworks.

Our research in cloud computing has expanded our knowledge of Microsoft Azure and GCP services, enabling us to integrate these services for seamless collaboration across modern technology stacks.

Behind the scenes, we refined server management skills, implemented Flask apps in Azure, established secure database connections, and implemented robust error handling techniques.

Also, we honed our time management skills through weekly sprint planning and participation in stand-alone events This approach has enabled us to better allocate our time, prioritize tasks and be more flexible at work of the evolving requirements, ensuring that we consistently deliver high quality results.

#### Accomplishments

One of our key developments is a fully functional web application that can be accessed on devices. The app accurately maps users' wardrobes, providing easy access from mobile and desktop platforms.

Application of material principles has become increasingly important to enhance safety and performance. Encapsulation ensured that data was maintained, while abstraction facilitated communication with the global object.

Our successful Azure SQL Database integration stands as a key achievement. To reduce redundancy and ensure data integrity, we developed a generalized database structure. Adding strong validation and constraints further enhances the accuracy of the data.

Our image processing capabilities, using REST APIs such as the Vision API, have enabled us to extract precise information from images. The use of the Jaccard Similarity coefficient reinforced keyword-based classification, exemplifying our commitment to enhancing the user experience by optimizing the ranking process.

# Conclusion

## Final Thoughts

Looking back, this journey has been truly rewarding. Not only did we gain a variety of skills that easily translate to real-world applications, including front-end and back-end development complemented by robust database management, but we also applied valuable insights to development in a rapid manner. In addition to technical skills, this experience highlights the importance of effective team collaboration, providing a deeper understanding of the dynamics that drive successful project outcomes. Reflecting on our accomplishments, we are excited about the journey ahead, equipped with new skills and perspectives that will surely inform our future endeavors.

Special thanks to the following: Advisor Professor A. Retik, Sponsors Krishna Langille and Pavan Sai Ram Korumilli, and head product testers Jacob Tea Lee and Kobe Kamin.

## Next Steps

Next steps to this project depend on resources, both fiscal and time, but in a world that is ideal and has infinite resources we would love to implement an AI powered recommendation algorithm to be a more useful application. We would also hope to add in the ability to have this application be on a mobile platform as it currently works in a mobile browser, but a mobile app would be better integrated. The final goal is likely to be wanting to have a fit predictor so you can visualize how your outfit looks not just on the screen but on oneself.