Deliverable – Final Product

1. Brief product description

"Hey, have you heard about **HokieFit**? It's this innovative platform we've created specifically for Virginia Tech students to revolutionize the way we share and engage with fashion. Picture this: you can post your outfits, tag them for different occasions, and get inspired by a diverse array of styles – all without revealing your identity. It's all about the outfits, not who's wearing them, thanks to our faceless posts policy and face detection tech. This means no bias, just pure fashion appreciation. Whether you're into vintage, casual, or professional looks, HokieFit is where your style finds its community. It's more than an app; it's a new wave in fashion, making it inclusive, judgment-free, and focused on creativity. Join us and be a part of this exciting fashion movement at Virginia Tech!"

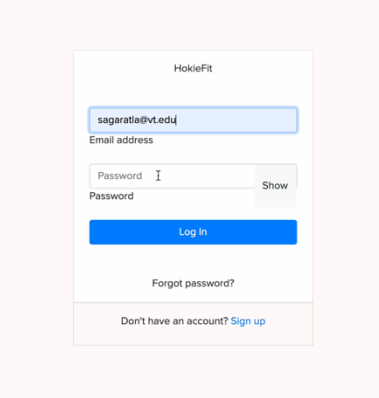
2. Product Functionalities

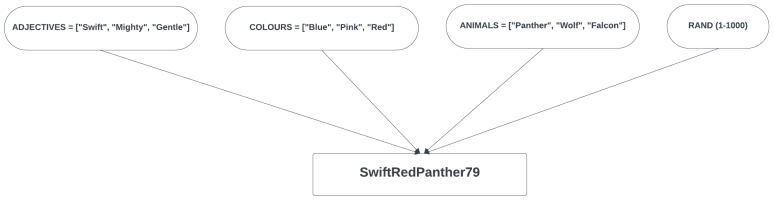
Below are all the major functionalities that were implemented as part of HokieFit.

2.1 Register / Login

- **Functionality**: This feature allows users to either sign up for a new account or access their existing account. The process is straightforward in any web application.

- **Highlights**: Unique to HokieFit, the registration includes an automatic generation of an anonymous username, enhancing user privacy and aligning with the app's anonymity focus. These usernames are creatively generated by randomly combining elements from predefined lists of adjectives, colors, and animals, along with a random number between 1 and 1000, ensuring unique and playful user identities while maintaining privacy.

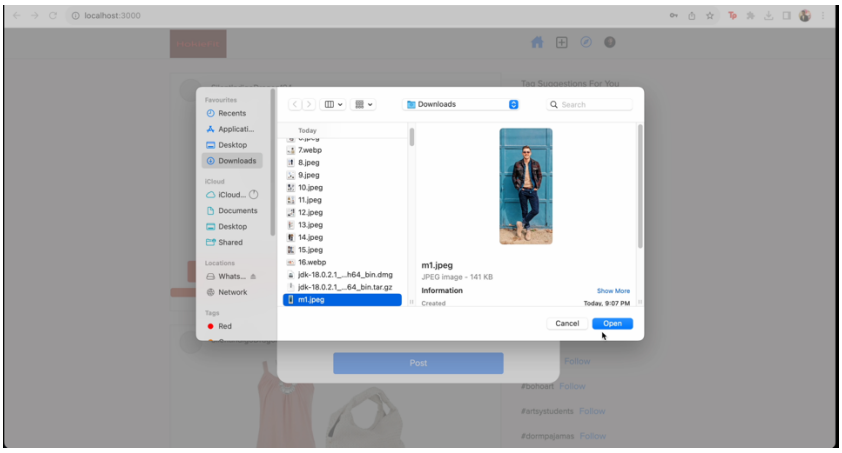




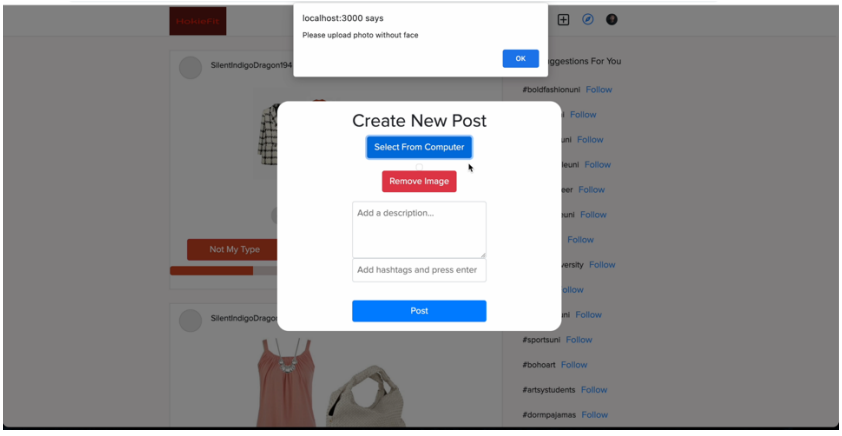
2.2 Post / Upload Outfits

- **Functionality**: Users can upload images of their outfits. The below screenshots depict the user interface while uploading outfits and error on uploading an outfit with a face.

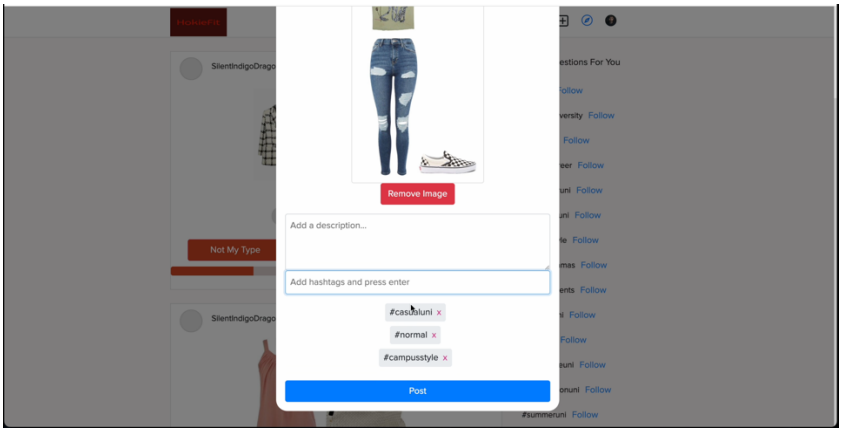
- **Highlights**: Outfit uploads incorporate advanced face detection to ensure user anonymity. Users can tag their outfits, automatically following these tags, a feature detailed in the sequence diagram.



Upload functionality



Error on uploading an image with a Face



Adding Tags while uploading Outfit

2.3 Follow / Unfollow Tags

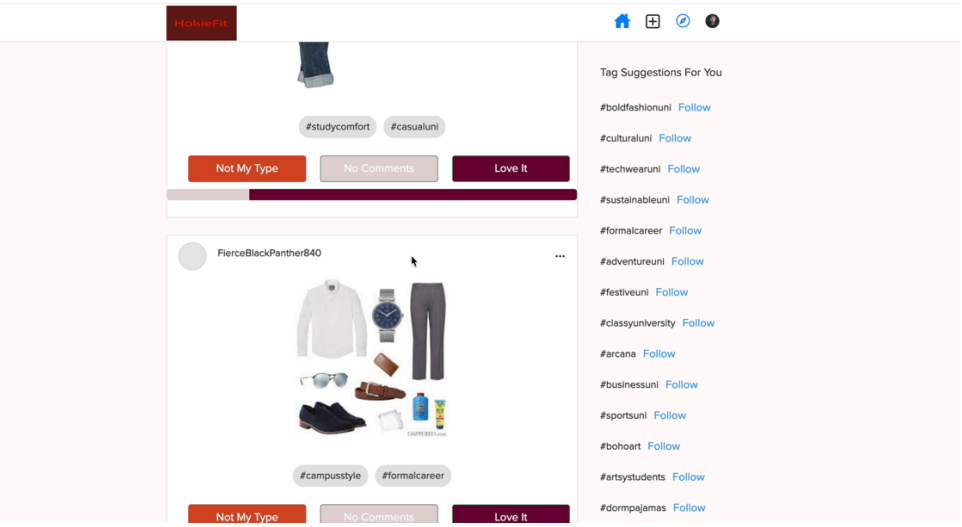
- **Functionality**: This allows users to tailor their HokieFit experience by choosing to follow or unfollow specific fashion tags. This feature can be seen in the upload functionality.

- **Highlights**: The tag following system is central to customizing content on the user's feed, enabling them to stay connected with preferred fashion trends.

2.4 Feed / Home Page

- **Functionality**: The feed page aggregates and displays outfits based on the user's followed tags.

- **Highlights**: Personalization is key here, as the feed is uniquely tailored to each user's fashion interests, promoting a more engaging and relevant browsing experience.

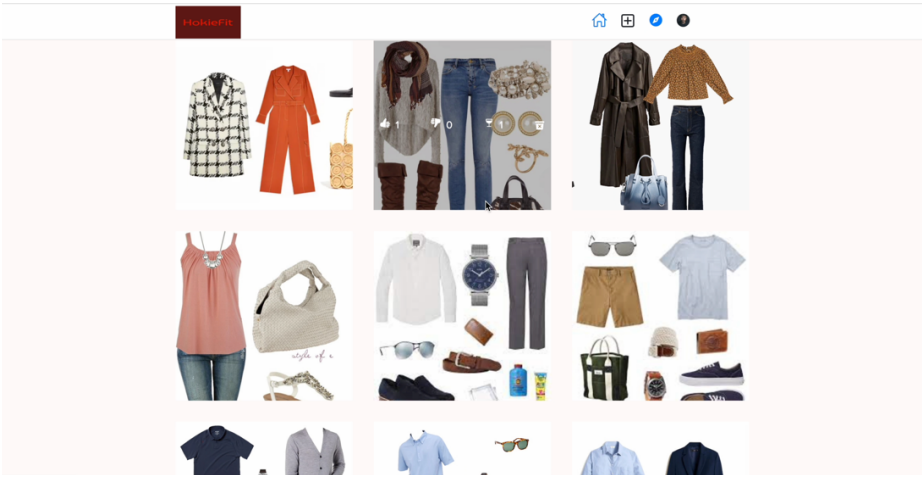


Feed Page showing Outfits & Tags that we can follow

2.5 Explore Page

- **Functionality**: Offers users a glimpse into fashion styles and tags they currently don't follow, encouraging exploration and discovery.

- **Highlights**: This feature is designed to broaden users' fashion horizons, introducing them to potential new interests outside their usual preferences.



2.6 Rate Outfits

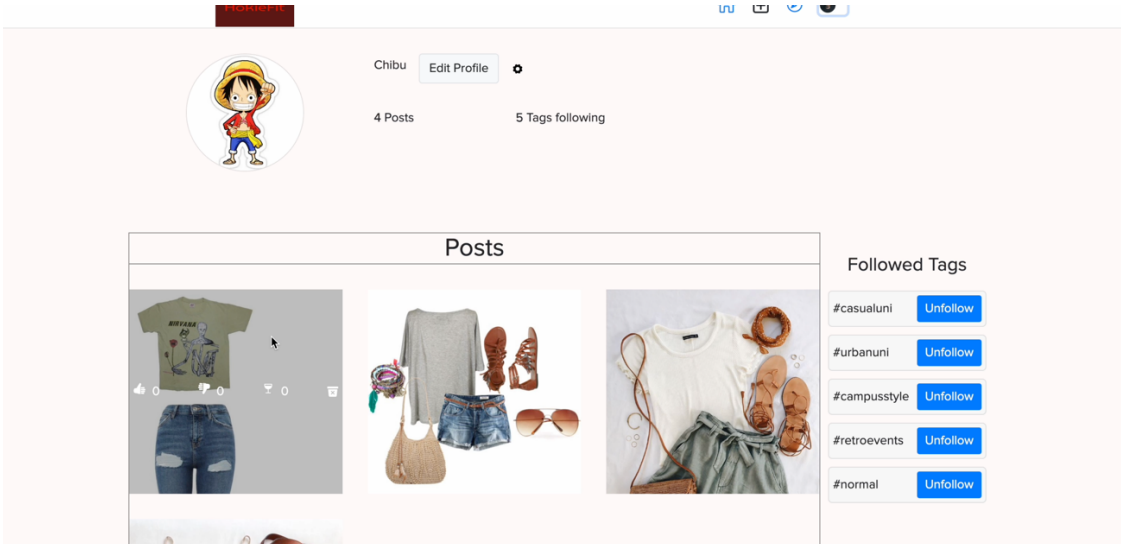
- **Functionality**: Users can interact with the community by rating outfits. The rating is visually represented by a horizontal bar divided into three color-coded sections. Each section's width is proportional to the number of votes it received, corresponding to the three rating options: Like, Neutral, and Dislike

- **Highlights**: The simple yet effective rating system (like, dislike, neutral) fosters community engagement and provides valuable feedback to others.

2.7 Profile Page

- **Functionality**: Enables users to view their own outfit posts and followed tags.

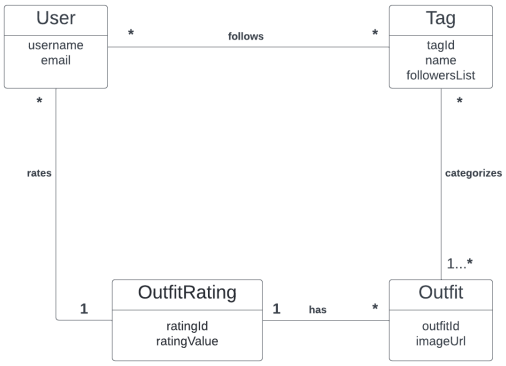
- **Highlights**: The profile page acts as a personal fashion diary, showcasing the user's own style contributions and their evolving fashion interests.



3. Design

3.1 Domain Model

In the HokieFit application, the interaction between users, outfits, tags, and ratings forms the core of the user experience. The system comprises four primary entities: User, Tag, Outfit, and OutfitRating, each playing a distinct role in the application's functionality.



User: This entity represents a person who uses the application.

- A User can *rate* zero or more Ratings, hence "*****" cardinal relationship.
- A User can *follow* zero or more Tags, hence "*****" cardinal relationship.

Tag: This is a label that can be followed by zero or more users and can categorize one or more outfits.

- A Tag can *categorize* zero or more Outfits, hence "*****"
- A tag can be *followed* by zero or more Users, hence "*****"

Outfit: An ensemble that can be categorized by one or more tags and can have zero or more ratings associated with it.

- Every outfit can have zero or more ratings, hence "*****"
- Every outfit can be *categorized* with at least one tag, hence "**1...***"

OutfitRating: A rating given by a user to an outfit. An outfit can have multiple ratings, but each rating is associated with exactly one outfit.

- Each OutfitRating is associated with one single user & single outfit, hence "**1**" cardinal relation with both.

The intricate interplay of these entities – User, Tag, Outfit, and OutfitRating – creates a dynamic and interactive platform where fashion is not just seen but actively rated, categorized, and followed, making HokieFit a vibrant and user-driven fashion community.

3.2 OpenCV Algorithm

Compilation of OpenCV Jar File:

The OpenCV Jar file with Java bindings was compiled from the OpenCV source code. This process involved configuring OpenCV with CMake, specifying options to enable Java bindings, and compiling the source using a compatible JDK version. The resulting Jar file and the associated native libraries (libopencv_java4xx.dylib or equivalent) are used in the application.

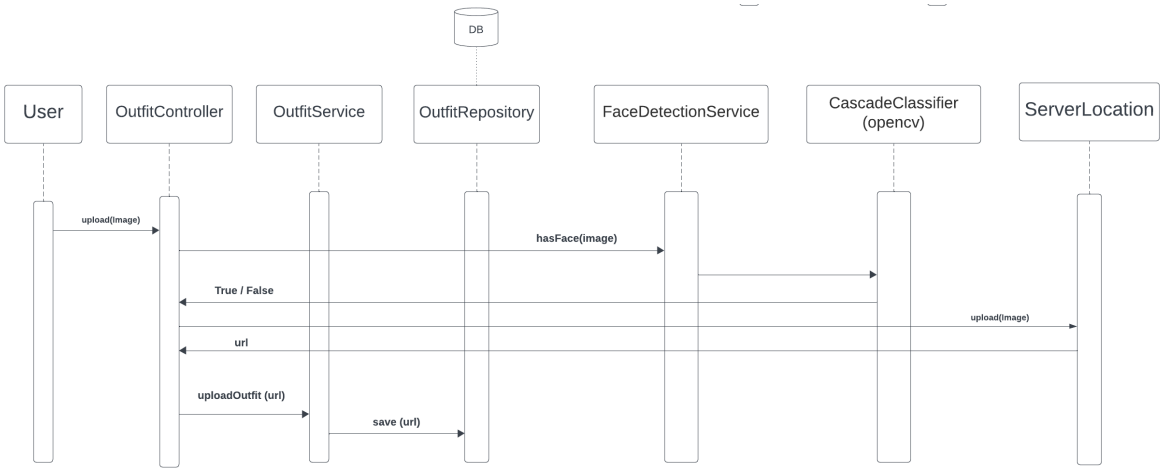
Haar Cascade Algorithm for Face Detection:

- ***Algorithm Origin***: The Haar Cascade is a machine learning object detection method used to identify objects in an image or video. It was proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.
- ***How It Works****:*
    - *Feature Selection*: The algorithm uses Haar features, which are simple digital image features resembling small parts of human faces (like the edge of the nose, the area around the eyes, etc.).
    - *Integral Images for Speed*: It utilizes a concept called integral images to quickly summarize the brightness of pixels within rectangular areas, allowing for the rapid calculation of Haar features.
    - *Cascade of Classifiers*: The most critical part of this algorithm is the 'cascade' of classifiers. It involves multiple stages where simple classifiers (decision trees) are applied to a region of interest and at each stage, irrelevant regions are discarded. If a region passes all stages, it is considered as a face.
    - *Training with Positive and Negative Images*: The classifiers are trained with numerous positive (images with faces) and negative (images without faces) samples.
- "***haarcascade_frontalface_alt.xml***" File:
    - This specific file is an XML representation of the trained cascade model for detecting frontal faces.
    - It's part of OpenCV's pre-trained models and is widely used for its efficiency and accuracy in detecting faces in various conditions.
- ***Usage in OpenCV:***
    - In OpenCV, this model can be loaded using the CascadeClassifier class.
    - Once loaded, it can be used to detect faces in images or video frames by calling the detectMultiScale method.

In summary, the "haarcascade_frontalface_alt.xml" is a powerful and efficient tool in OpenCV for frontal face detection, leveraging the speed and simplicity of Haar Cascade classifiers.

3.3 Interaction Diagrams

3.3.1 Interaction / Sequence Diagrams – Upload Outfit-



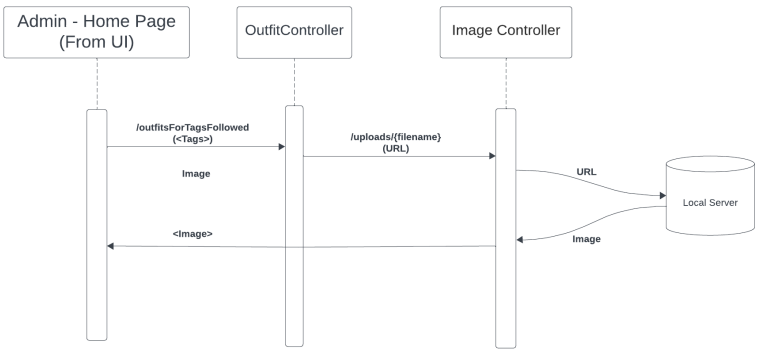
A Peek at the Image Detection Flow

- The user first uploads an image via the REST API endpoint.
- Instead of directly uploading the outfit, we make a call to the face detection service.
- OpenCV's **Imgcodecs** class is used to load the uploaded image into memory (**Imgcodecs.imread**).
- The **CascadeClassifier** class in OpenCV is initialized with the Haarcascade XML file (**haarcascade_frontalface_alt.xml**). This file contains the pre-trained model for face detection.
- The **detectMultiScale** method of **CascadeClassifier** is used to detect faces in the image.
- Based on the output of the face detection classifier, we decide whether the user can upload the outfit or not

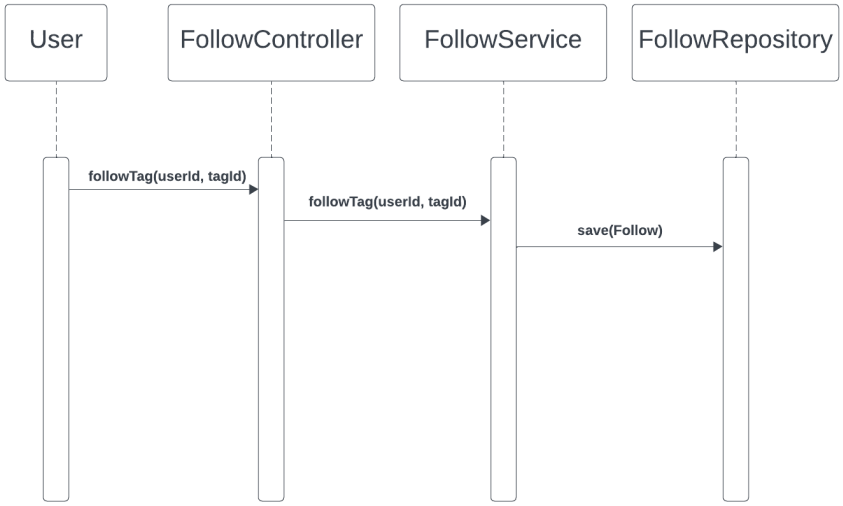
3.3.2 Interaction / Sequence Diagrams – Upload Outfit Retrieval Part

This was added as part of bug fixing the retrieval of image after uploading from the front-end process.

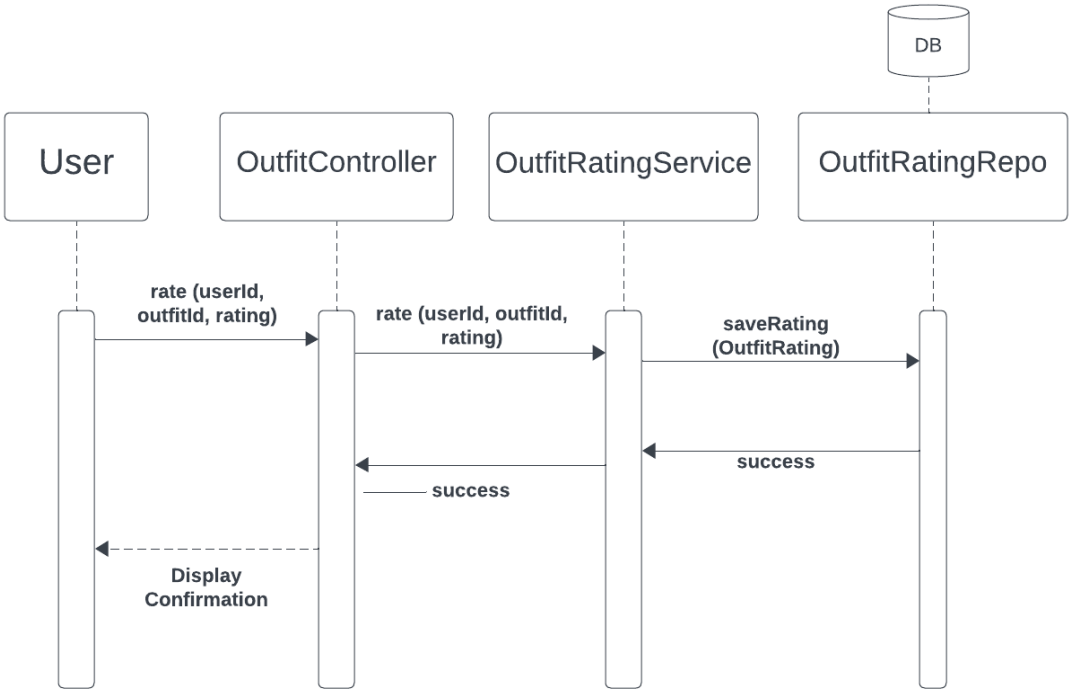
- After uploading, user is redirected to home page where all the outfits that he follows are shown
- Now, we hit the new API written in Image Controller which retrieves from the URL of the image passed



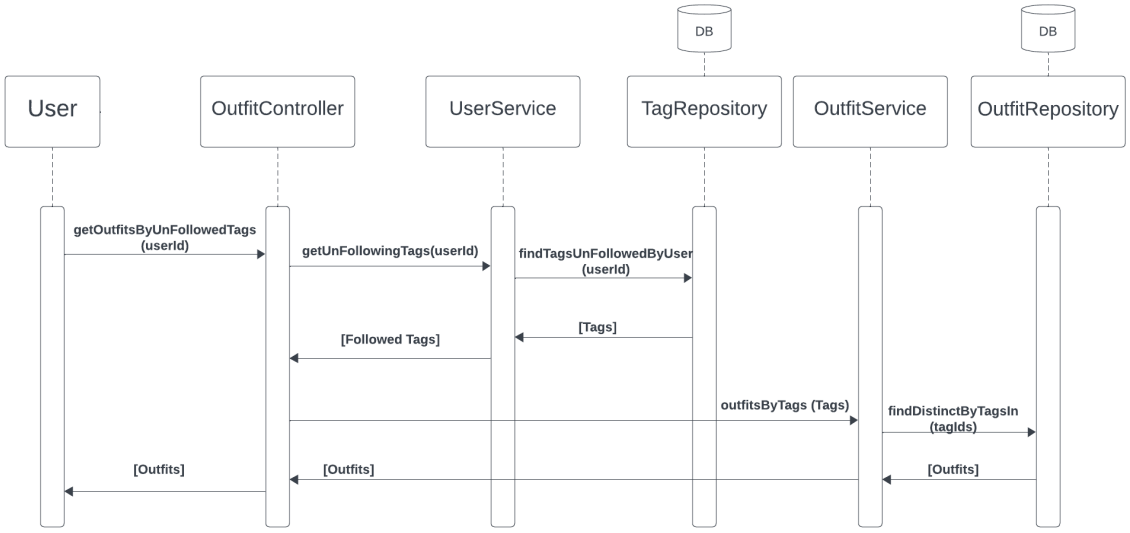
3.3.3 Interaction / Sequence Diagrams – Follow / Unfollow Tag



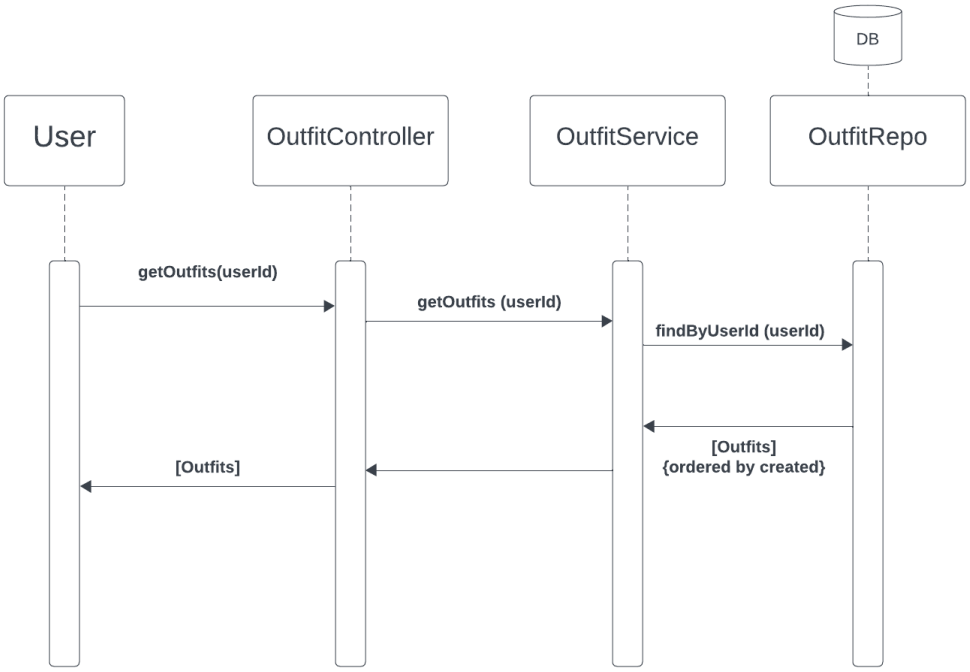
3.3.4 Interaction / Sequence Diagrams – *Rate Outfit*



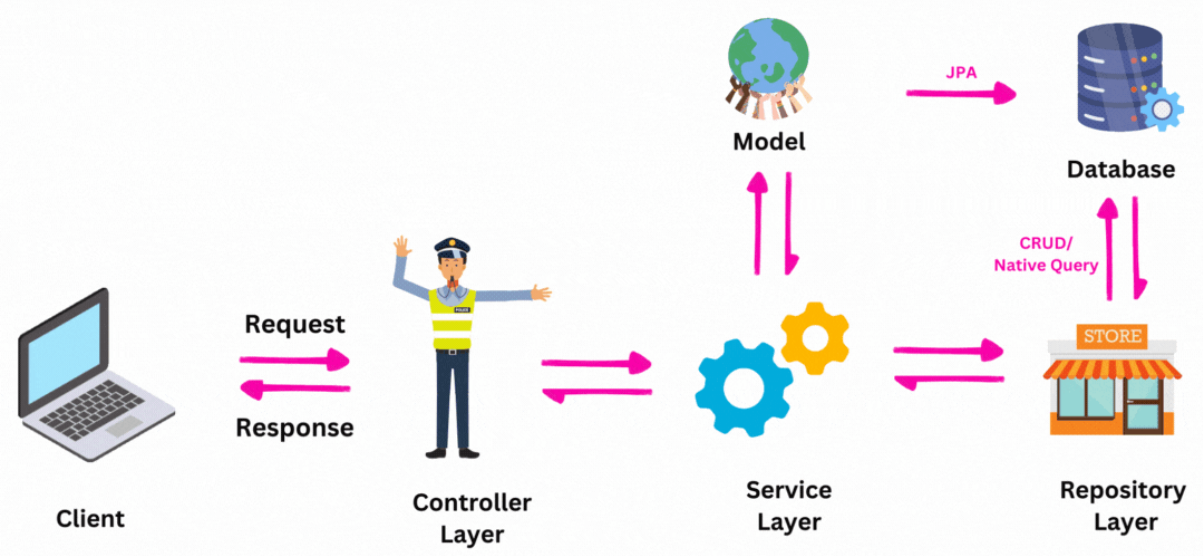
3.3.5 Interaction / Sequence Diagrams – Explore Page



3.3.6 Interaction / Sequence Diagrams – Profile Page

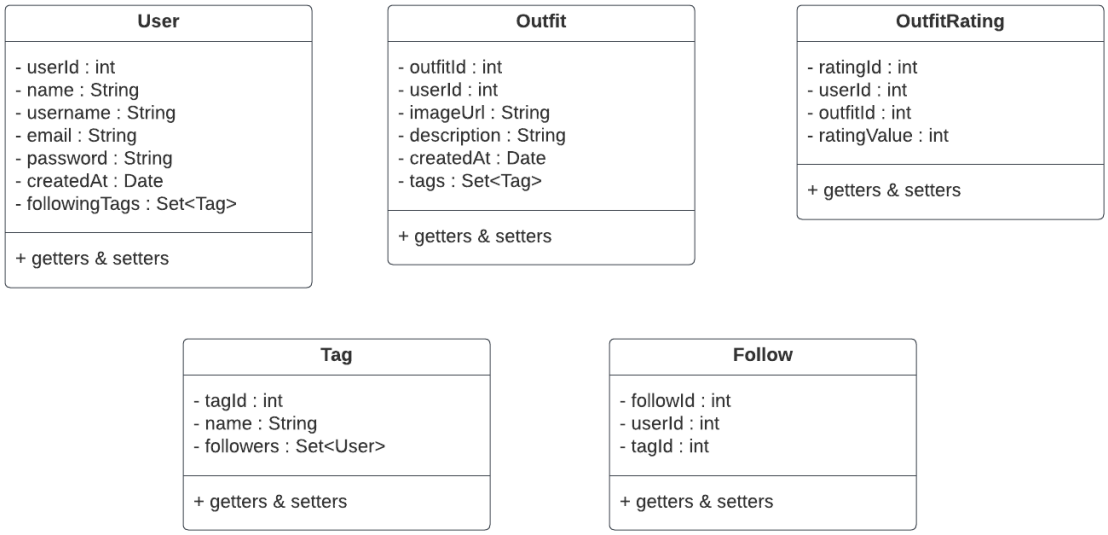


3.4 Class Diagrams-

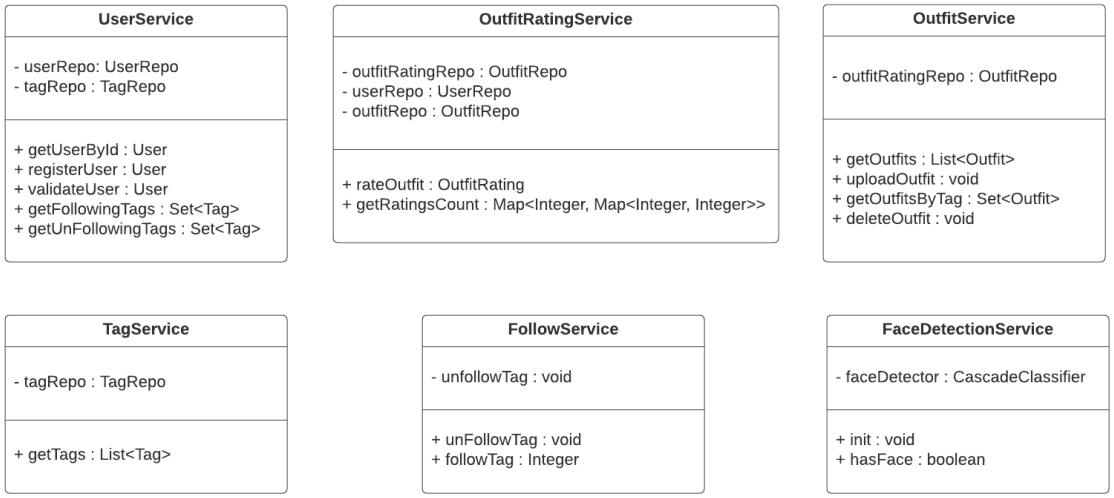


- This architecture primarily drives the notion of class diagrams behind our project.
- Like the above image, our classes are bifurcated into Model, Controller, Service & Repository

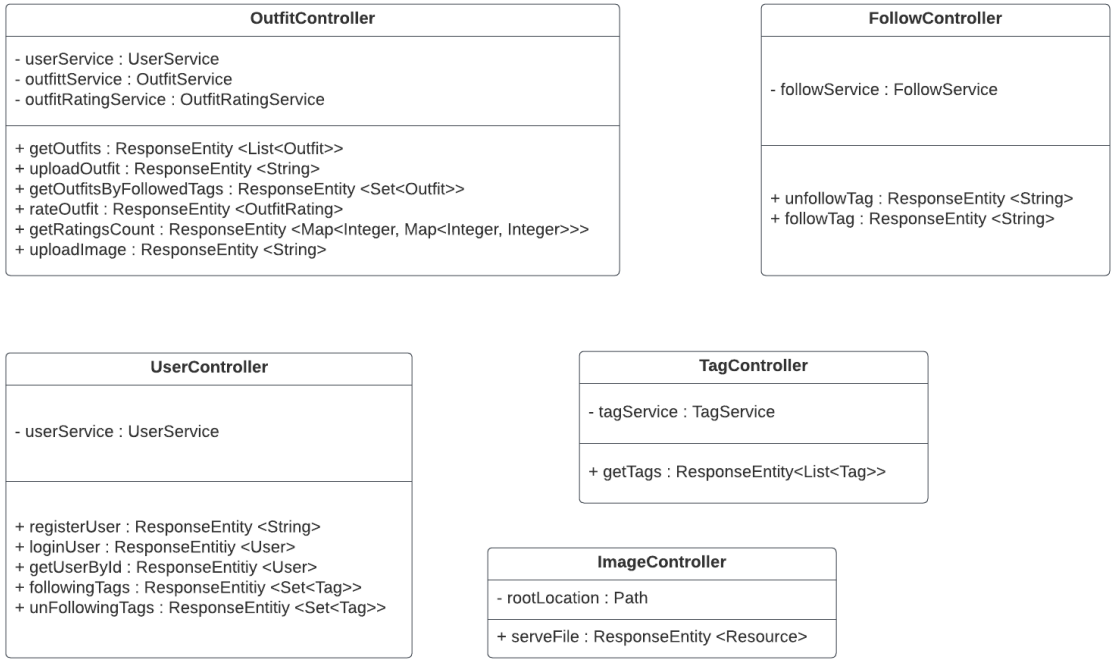
3.4.1 Model Layer



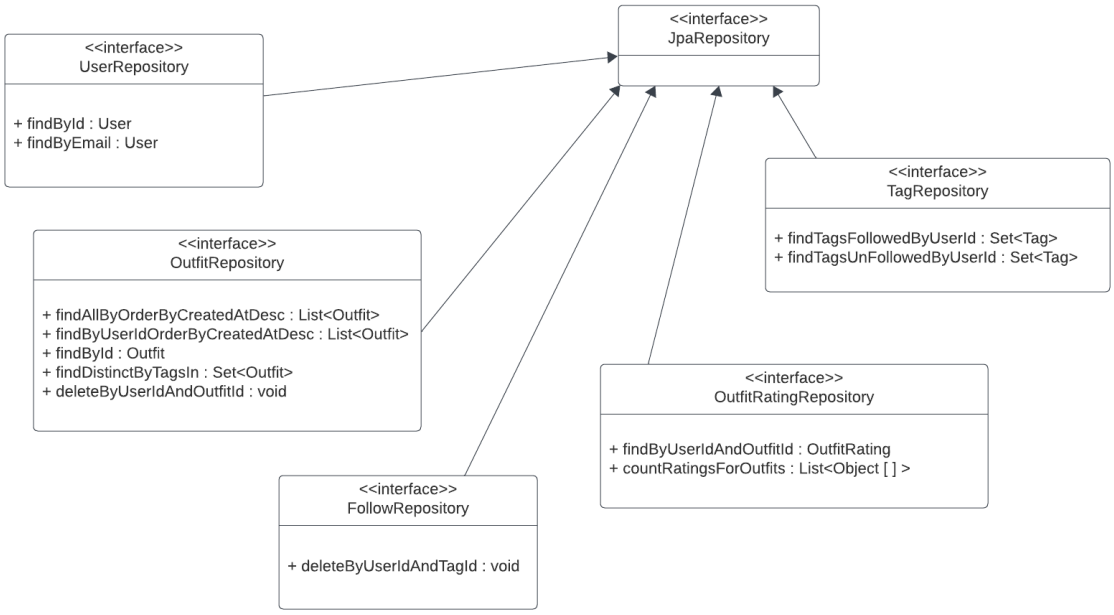
3.4.2 Service Layer



3.4.3 Controller Layer



3.4.4 Repository Layer



3.5 Tools Adopted and Product Uniqueness

3.5.2 Uniqueness of HokieFit

1. **IntelliJ IDEA for Java Development (Spring Boot)**:
    a. Choice for Backend Development: IntelliJ IDEA was chosen for its robust functionality, intelligent coding assistance, and comprehensive support for Java development, making it ideal for working with Spring Boot.
    b. Spring Boot: This framework was selected for the backend to create stand-alone, production-grade Spring-based applications with ease. Spring Boot's auto-configuration and embedded server support significantly accelerated the development process.
2. **Visual Studio Code for Frontend Development (React):**
    a. Frontend Development Environment: VS Code, known for its lightweight, yet powerful features, was used for developing the frontend in React. It offers a user-friendly interface, excellent extension support, and effective integration with various development tools.
    b. React Framework: Chosen for its efficiency in building user interfaces, React.js allows for creating dynamic and responsive web pages. Its component-based architecture facilitated the modular development of HokieFit user interface.
3. **MySQL for Database:**
    a. Relational Database Management System: MySQL, a widely used open-source relational database, was employed for its reliability, performance, and ease of use. It provided a solid foundation for managing user data, outfit information, tags, and ratings.

3.5.2 Uniqueness of HokieFit

1. **No Faces, Just Fashion***:*

HokieFit introduces a unique approach to fashion sharing by ensuring that no faces are shown in outfit posts. This policy encourages users to focus on the fashion and style aspects rather than the person's appearance or beauty, fostering a bias-free environment.

2. **Anonymous Usernames for Enhanced Privacy***:*

   In HokieFit, users don't use personal usernames. Instead, the system generates anonymous usernames, enhancing user privacy and further aligning with the app's commitment to reducing biases and ensuring that attention remains solely on fashion content.

3. **Following Tags Instead of People***:*

   Content-Centric Interaction which allows breaking away from traditional social media norms, HokieFit allows users to follow tags rather than individual accounts. This unique feature shifts the focus from personal popularity to the quality and relevance of fashion content, promoting a more content-driven and egalitarian platform.

In summary, HokieFit stands out with its innovative approach to fashion sharing, enabled by a robust technology stack including IntelliJ IDEA, Visual Studio Code, React, Spring Boot, and MySQL. The platform's distinctive features like faceless outfit posts, anonymous usernames, and the emphasis on following tags over people establish it as a unique player in the digital fashion space, promoting a more inclusive, unbiased, and content-focused fashion community.

4. Retrospective

4.3.1 What Went Well

- **Effective Team Collaboration:** Throughout the sprints, our team maintained strong communication and collaboration, which was crucial in navigating challenges and achieving milestones.
- **Environment Setup and Feature Implementation:** The development environment was established smoothly, and key features like user registration, login, and feed page were successfully implemented.
- **Adaptability to Technical Challenges:** The team showed resilience in overcoming technical issues, such as integrating OpenCV and handling image uploads.

4.3.2 What went wrong

- **Incomplete Features**: Certain functionalities, like outfit upload UI and Explore/Search UI, remained incomplete, impacting the overall user experience.
- **Challenges in Frontend Development:** The front-end development initially lagged, affecting the project's pace.
- **Underestimation of Tasks:** Initial task estimations were off mark, leading to delays and backlog in subsequent sprints.

What have we learned by working on this product-

Importance of Accurate Task Estimation: We learned that precise task estimation is vital for timely delivery and avoiding sprint spillovers.

Value of Testing: The necessity of incorporating automated testing early in the development process became apparent, emphasizing its role in maintaining quality.

Technological Preparedness: The project highlighted the importance of being technically prepared, especially when dealing with advanced features like OpenCV.

What Actions could have been taken to Improve the product-

Enhanced Focus on UI Development: Prioritizing UI development and ensuring parity with backend progress could have enhanced the product's user experience.

Implementing Testing Sooner: Incorporating testing/unit testing earlier would have helped identify and rectify issues swiftly, ensuring a smoother development process.

More Resources for Challenging Features: Allocating additional resources and time to technically challenging features, like image upload, could have prevented delays.

How to make this product better in future-

Continuous Learning and Skill Enhancement: Our team should continue to upgrade their skills, particularly in front-end technologies and image processing algorithms. This will ensure that our product remains technologically advanced and relevant to market needs.

Real-Time Face Detection Improvement: Enhancing the face detection feature to perform in real-time will significantly improve user experience. We should focus on optimizing the algorithm for speed and accuracy without the need to store user-uploaded images, thus enhancing privacy and efficiency.

Handling Load and Scalability: As the user base grows, we need to ensure that our application can handle increased loads without compromising performance. Implementing scalable architecture and efficient load balancing techniques will be crucial.

Balanced Resource Allocation: A well-balanced allocation of resources across UI/UX design, backend development, testing, and other project areas is essential. This balance will help maintain a steady rate of progress and ensure that all aspects of the project receive the attention they need.

Deploying and Continuous Integration: Setting up a robust deployment process with continuous integration will help in quick and efficient roll-outs of new features and updates. This will also aid in early detection and resolution of any issues.

Addressing Security Concerns: As we handle user data, particularly images, prioritizing security in every aspect of the application is crucial. Regular security audits and updates should be part of our development cycle.

5. Contributions

Sagar Atla

- Core Backend: Vital role in designing and developing the backend infrastructure, including API endpoints and database schemas. Implemented outfit upload functionality and outfit deletion logic.
- Quality Assurance: Contributed significantly to testing infrastructure decisions and quality checks.
- Special Contributions: Pioneered the integration of OpenCV for face detection, developed the anonymous username generation algorithm, and actively participated in report writing and presentation preparation.

Mohith Kamanuru

- Core Backend: Contributed to backend development, especially in the implementation of feed logic and outfit rating features.
- Quality Assurance: Participated in quality assurance, focusing on performance and reliability.
- Special Contributions: Took the lead in integrating and fixing the upload outfit functionality and developed auto-follow tags feature.

Aruj Nayak

- Core Backend: Assisted in backend development, including setting up GitLab and Scrum boards for project management.
- Quality Assurance: Involved in testing and ensuring code quality.
- Special Contributions: Managed integration testing, usability testing, and prepared dummy data for integration testing. Also responsible for designing and implementing the homepage with outfit tags.

Srikanth Karri

- Core Frontend: Key player in front-end development, focusing on UI design and integration, particularly for the profile page and tag display.
- Quality Assurance: Engaged in quality assurance processes.
- Special Contributions: Designed search functionality UI and integrated search API with the frontend. Also played a significant role in presentation preparation.

Sai Pavan Bathala

- Core Frontend: Pivotal in developing the front-end architecture, including designing visually appealing transitions and animations.

- Quality Assurance: Actively involved in quality checks and performance testing.
- Special Contributions: Led the UI design for the "Feed" page, implemented follow/unfollow tag functionality, and conducted research on potential frameworks and technologies. Also responsible for the UI integration for outfit upload and face detection.

Each team member has made significant contributions to the project, demonstrating a blend of technical expertise, collaborative spirit, and commitment. From backend and frontend development to quality assurance and deployment, their collective efforts have been instrumental in driving the project towards its goals. The project also benefited from their additional roles in areas like research, testing, report writing, and presentation preparation.