dripdrop

Design Document

Team 25

Kaden Wingert, Kolby Kucera, Elyse Kriegel, Logan Roe, Zachary Foote, Gavin Rich

Sdmay25-25@iastate.edu https://sdmay25-25.sd.ece.iastate.edu/

Executive Summary

Presently, in social media outfit sharing, apparel company affiliates typically must have 10,000+ followers before companies approve them to be registered affiliates. With our solution *dripdrop*, this is no longer the only path to affiliate apparel sharing. *dripdrop* is an apparel-sharing social media app that allows the everyday user to not only share their favorite outfits but also to view outfits and find the best deals on the looks they desire. The idea is that our company will have large-scale affiliate partnerships with many of the leading apparel companies so that the users of our app can earn us a commission, and we will trickle most of this money down to the users. Additionally, to save people money on the posting side, when users see post outfits they like, we will have an AI-generated list that the user can click into, which will show items with similar attributes from different sites so that the user can still get the look but for the best price.

The key design requirements at the highest level will be a working website, a working mobile application, and a working AI-outfit matching model. We want a smooth and intuitive user interface for the website and the application that provides the user with a simple and convenient experience. The goal is an application where users can create an account, follow other users, like and comment on posts, create posts for themselves, search accounts, and save posts.

For our design, we decided that for the database, we are using a MySQL database and an S2 bucket both hosted on Amazon Web Services. We use Python to create endpoint lambdas for the backend to ensure our app is efficient and cost-effective. For the front end, we will be using React, as it allows for universal code between the website and the mobile application.

Thus far, our team has set up the database and the website to be hosted on AWS, started creating a model that can determine different qualities of an apparel item and create tags for the qualities, made most of the lambdas for the backend, and have a majority of the pages with their basic functionality on the frontend. Our website meets some user needs presently, as the page is simple, intuitive, andsic functionality. Still, improvements must be made, including faster loading times for the feed, a better way to search profiles, and a better feed algorithm that can personalize the user's experience.

Our next steps are to transition to creating the mobile application, as only the website has been worked on thus far. Additionally, we will work on finishing the AI-outfit matching model for the app and polishing up the front so that pages are correctly formatted. All user functionality (following, liking, commenting, saving posts) is added and works correctly.

Learning Summary

Development Standards & Practices

- Standards:
    - **IEEE 1448a-1996** - Standard for Information Technology - Software Life Cycle Processes
    - **IEEE/ISO/IEC 12207-2017** - International Standard - Systems and software engineering -- Software life cycle processes
    - **IEEE 1012-1998** - IEEE Standard for Software Verification and Validation
    - **IEEE/ISO/IEC 29119-2-2021** - ISO/IEC/IEEE International Standard - Software and systems engineering - Software testing -- Part 2: Test processes
    - **IEEE 1016-1987** - IEEE Recommended Practice for Software Design Descriptions
    - **IEEE/ISO/IEC 42010-2022** - IEEE/ISO/IEC International Standard for Software, systems, and enterprise--Architecture description
- Practices:
    - Standard naming conventions for Python and typescript
    - Followed AWS best practices where applicable
    - HTTP status code guidelines for API responses
    - Concise and consistent comments
    - CRUD model for backend code

Summary of Requirements

- Fully functioning frontend: Website, IOS app, Android app
    - User authentication
    - Posts
    - User profile and feed
    - AI Price comparison feature
- Backend
    - MySQL Database
    - API Endpoints
    - Backend CRUD operation and business logic functions
    - Storing images
- Infrastructure
    - AWS CDK code to setup/manage our entire infrastructure
    - Host database
    - Host frontend code
    - CI/CD pipeline
- AI
    - Obtain comprehensive dataset
    - Create or obtain a model
    - Train model

Applicable Courses from Iowa State University Curriculum

- COM S 309
- COM S 363
- SE 319

- COM S 228
- SE 422
- SE 409

New Skills/Knowledge acquired that was not taught in courses:

- AWS Infrastructure
- AWS CDK Code
- AWS Services: Lambda, RDS, API Gateway, Aurora, Route 53, Secrets Manager, EC2 Instance
- Python
- Typescript
- React Native
- Creating and training our own AI model
- Bruno - API client

Table of Contents

Definitions and Key Concepts

1. **AI Tag Matching**: The process of matching clothing attributes (tags) AI models generate to suggest similar products.
2. **Amazon Aurora**: A relational database service offering high performance and scalability.
3. **AWS CDK**: AWS Cloud Development Kit, a framework to define cloud infrastructure using code.
4. **AWS CloudFront**: A content delivery network (CDN) service that caches and serves static files and images globally, reducing latency for users in different regions.
5. **AWS S3**: Amazon Simple Storage Service, used for storing and retrieving media files and hosting static website components with high availability and scalability.
6. **AWS-Hosted Server**: An AWS-hosted server is a virtual server hosted in the Amazon Web Services (AWS) cloud infrastructure. It provides scalable and reliable computing resources for deploying and running applications, such as hosting backend services and databases for web applications.
7. **Budget-Conscious Shopper**: A user demographic focused on minimizing expenditures while maintaining fashionable looks. Often includes students, young professionals, or families who prioritize affordability.
8. **Bruno**: A testing tool used for verifying API behavior by testing endpoints for expected request and response patterns. It simplifies API testing by allowing users to automate test cases.
9. **CI/CD Pipeline**: Continuous Integration and Continuous Deployment—a methodology for automating code testing and deployment.
10. **CloudWatch**: An AWS monitoring and observability service that provides actionable insights into applications and infrastructure. It collects and visualizes metrics, logs, and traces to help developers monitor application performance and troubleshoot issues.
11. **CRUD**: Create, Read, Update, Delete (operations for database and API).
12. **Dynamic Test Processes (IEEE/ISO/IEC 29119-2-2021):** A systematic testing approach that includes test design, setup, and execution. It ensures comprehensive evaluation of software requirements and performance.
13. **Empathy Mapping**: A collaborative tool to understand users' thoughts, feelings, and needs, ensuring the solution addresses real-world problems effectively.

14. **Fast Fashion**: A clothing industry characterized by rapidly produced, low-cost garments to meet current trends and often criticized for contributing to unsustainable shopping habits.
15. **Fashion Enthusiast**: An individual passionate about staying updated with the latest fashion trends, seeking to replicate styles seen on social media or through influencers.
16. **IAM Policies**: Identity and Access Management (IAM) policies are rules that define permissions for actions and resources in AWS. They ensure that resources are accessible only to authorized users or systems.
17. **Image Recognition**: An AI-driven model analyzes uploaded clothing images and suggests descriptive tags (e.g., "denim jacket").
18. **Kanban Board**: A task management system that uses visual columns to track the status of tasks.
19. **Lambdas**: In AWS, lambdas refer to AWS Lambda functions. These serverless computing functions allow users to run code in response to events and manage the computing resources automatically without needing to provision or manage servers. A benefit of lambdas is that you are only billed for the time that the function runs, so you don't pay for unnecessary idle server time.
20. **Modern UI/UX Standards**: Design principles emphasize simplicity, clarity, and responsiveness, ensuring an intuitive and engaging user experience.
21. **Price Comparison**: A tool or feature that allows users to analyze pricing across multiple retailers to identify the most cost-effective options for a product.
22. **Retail APIs**: External application programming interfaces are provided by retailers (e.g., Amazon, eBay, Walmart) to fetch product details like pricing, availability, and alternatives.
23. **Role-Based Access Control (RBAC):** A system that restricts access to specific actions or data based on defined user roles, ensuring security and proper permissions management.
24. **Route 53**: A scalable DNS web service provided by AWS for managing domain names, ensuring traffic is routed effectively to AWS resources or external endpoints.
25. **RPS**: Responses per second (API throughput metric).
26. **Social Media Influencer**: A content creator on platforms like Instagram or TikTok who showcases fashion-related material. They often engage their audience by sharing outfit ideas, trends, and direct links to purchase items.
27. **Static Website Hosting**: A method of hosting static content, such as HTML, CSS, and JavaScript files, without requiring a backend server. AWS S3 and CloudFront are often used for this purpose to ensure scalability and low latency.
28. **Unit Testing**: Testing individual software components to ensure they perform as expected.
29. **User Personas**: Fictional representations of target users based on demographics, needs, and behaviors to guide design and development.
30. **Verification and Validation** (IEEE 1012-1998):
    a. Verification: Ensures the software is developed correctly according to specifications (e.g., through code reviews and automated tests).

b. Validation: Confirms that the final software product meets user requirements and performs as intended.

1. Introduction

1.1 Problem Statement

With the growing popularity of social media, consumers' clothing choices are constantly influenced by online fashion trends. However, finding affordable versions of popular clothing articles is challenging, particularly for users who don't have the time or desire to search across multiple platforms and retailers. The fashion industry often markets expensive clothing that is out of reach for the average consumer, creating a divide. Thus, budget constraints make individuals feel disconnected from the trends they love.

Our solution, dripdrop, aims to improve the accessibility for users in the clothing industry. While social media allows users to engage with fashion content, it doesn't offer the tools to make informed purchasing decisions. Our solution will help users discover where to buy the clothes they see in posts while providing cheaper alternatives to ensure affordability. This approach addresses the issue of rising clothing costs and the lack of transparency of prices for fashion. Dripdrop targets users who want wardrobe inspiration without overspending, giving them the resources to shop smarter. We are promoting affordability and accessibility in fashion by providing a platform where users can upload images of clothing, link to retailers, and view price comparisons. This is meant to encourage more cost-effective and thoughtful purchasing decisions, addressing a global issue where fast fashion contributes to unsustainable shopping habits.

1.2 Intended Users

1. Fashion Enthusiasts

- **Description**: Fashion enthusiasts are individuals who have a desire to stay up-to-date with the latest trends. They pay attention to influencers and their styles and search for ways to replicate these looks. The age range of these users ranges from teens to adults.
- **Needs**: Fashion enthusiasts need a single platform to quickly find the clothing they see online and identify budget-friendly alternatives. They value style and affordability, making it essential to access diverse retailers and pricing options in one place.
- **Benefit**: Our platform lets fashion enthusiasts quickly locate popular clothing items and purchase affordable alternatives. Thus, the time and effort required to find desired, affordable outfits is reduced. The platform will help fashion enthusiasts make informed decisions, enhancing their shopping experience.

2. Budget-Conscious Shoppers

- **Description**: This group includes individuals whose main goal is to minimize spending money when making purchases, particularly students, young professionals, or families. These users want to stay fashionable but prioritize keeping their expenses low.
- **Needs**: Budget-conscious shoppers need access to price comparisons and cheaper alternatives when shopping for clothing. They are looking for ways to achieve stylish outfits without spending too much.
- **Benefit**: Our platform assists budget-conscious shoppers in their efforts to save money by displaying the lowest prices for clothing items and offering similar, more affordable alternatives. This feature directly addresses their desire for affordable fashion while ensuring they can access styles that fit within their budget.

3. Social Media Influencers

- **Description**: Social media influencers are content creators who post fashion-related material on social media, such as Instagram or TikTok. They maintain engagement with their audience through showcasing outfits and fashion trends. Influencers often receive questions asking where the featured items in their posts can be purchased.
- **Needs**: Influencers need a simple way to share details about their clothing and recommend budget-friendly options to their followers. They also benefit from tools that allow them to link to retailers and compare prices seamlessly.
- **Benefit**: Our platform enables influencers to post outfits with direct links to the most affordable options. By offering this feature, they can engage their audience more effectively and provide added value by promoting budget-friendly fashion. This helps increase their credibility and maintain a strong connection with their followers, who appreciate the simplicity of finding affordable options.

4. Retailers and Brands

- **Description**: Retailers and brands are businesses and vendors that sell clothing and accessories. They desire to reach wider audiences and connect with potential buyers by leveraging online platforms for marketing and sales.
- **Needs**: Retailers need visibility for their products, especially when it comes to competing with other brands in a crowded market. They also need insights into consumer trends and buying habits.
- **Benefit**: Retailers benefit from our platform, displaying their products to users who are actively looking for affordable clothing options. They can drive sales by offering competitive prices and reaching budget-conscious consumers who may not have otherwise considered their brand.

Empathy Mapping and Personas FigJam Board:
https://www.figma.com/board/NaHKgQEzDSHwqSdj0KS3mO/sdmay25-25?node-id=0-1&node-type=canvas&t=k3Vj84AyTggf38WO-0

2. Requirements, Constraints, And Standards

2.1 Requirements and Constraints

2.1.1 Functional Requirements

1. The app must require all users to sign up via email or social media accounts.
2. The app must allow all users to create and update their profile with basic information such as name, profile picture, and bio.
3. The app must allow all users to follow others to see their clothing posts and like/comment on them.
4. Users need the ability to upload images of their clothing with descriptions (brand, price, size, etc.)
5. The app must automatically suggest tags based on image recognition (e.g., "denim jacket," "red shoes").
6. The system must use AWS S3 to handle user-uploaded media.
7. The system must use AWS S3 for static website hosting.
8. The system must leverage AWS SNS to send price drop notifications to subscribers.
9. Users must be able to reset their password using their email address.
10. The system must leverage AI to recommend similar clothing items based on an image of a clothing item.
11. Users must be able to filter clothing items by price, brand, and clothing article.

2.1.2 Resource Requirements

1. The product must allow users can search for products by name, brand, or category (e.g., "Nike sneakers").
2. The product must implement advanced filters (size, color, price range, brand, etc.).
3. When a user clicks the save ribbon, the product or outfits they like will be added to their wishlist for future reference.
4. When a user's wishlist product has dropped in price or is running out of stock, the app must notify the user.
5. The system must use Amazon Aurora as a scalable database.
6. The database must support replication across multiple availability zones (AZs) to ensure high availability and fault tolerance.
7. Implement role-based access control (RBAC) to define user roles and permissions, ensuring restricted access to sensitive actions or data.
8. The system must use AWS CloudFront to cache and serve content (static files and images) globally, reducing latency for users in different regions.
9. Use GitLab actions for continuous integration and deployment, automating the build, test, and deployment processes for rapid iteration.
10. Use AWS CDK to provision the cloud infrastructure.
11. Use AWS Cost Explorer to track and monitor AWS resource costs, ensuring that the system remains within budget.

2.1.3 Aesthetic Requirements

1. The system must integrate with external retail APIs (e.g., Amazon, eBay, Walmart) to fetch product details, prices, availability, and alternatives.
2. The system must dynamically scale API requests based on user volume.
3. The application must be navigable and intuitive enough for any new users to create a post in under 90 seconds.
4. The app must offer a dark mode option for users, switching between light and dark themes based on user preference.
5. The system must use legible, modern fonts with proper size and spacing to ensure content is easy to read on all screen sizes.

2.1.4 User Experiential Requirements

1. The product must adhere to modern UI/UX standards.
2. To enhance user navigation, the system must use straightforward typography for optimal readability, with a defined font hierarchy (headers, subheaders, body text).
3. The social feed should emphasize simplicity and clarity, similar to leading social media platforms (e.g., Instagram).
4. The product must provide visual feedback (e.g., color changes, shadows) for interactive elements such as buttons, links, and images when hovered or clicked.

5. The system must follow a consistent design language across all pages, including consistent use of color schemes, typography, button styles, and spacing.
6. When images or data are being fetched from the server, the app will provide progress indicators to inform users.
7. The app must use small animations or color changes for actions like liking a post, saving a wishlist item, or clicking buttons to give the user instant feedback.

2.2 Engineering Standards

2.2.1 Importance of Engineering Standards

Engineering standards are essential because they help ensure that different products and technologies work together smoothly. Engineering standards are agreed-upon rules that guide how devices are made and used, making life easier for everyone. For example, a Bluetooth speaker can seamlessly connect to any type of smartphone, thanks to standards.

In addition to making things integrate well, standards define aspects of the production, installation, and use of technology that ensure products are safe, reliable, and built to last. By following these guidelines, companies and engineers don’t have to reinvent ideas whenever they create something new. Instead, they can follow proven practices others have used. This leads to a level of consistency and reliability across the industry.

Standards help push innovation forward, creating a foundation for new technologies. When everyone follows the same set of rules, it’s easier to build on existing ideas and create new + exciting products that people can use confidently.

2.2.2 IEEE Standards applicable to dripdrop

The software's life cycle is one of the project's most essential portions. IEEE 1448a-1996, "Standard for Information Technology - Software Life Cycle Processes, " is an IEEE standard that covers this.” Firstly, the standard discusses the software life cycle phases, including essential steps such as planning, analyzing requirements, implementation, testing, and deployment. Beyond the software life cycle phases, the standard also speaks of different categories for processes, such as primary, supporting, and organizational processes. Primary processes involve activities that directly impact software production, such as development. Supporting processes include activities that support primary processes, such as documentation or management. Lastly, organizational processes mean activities such as project management. The standard also speaks on quality and consistency with a clear emphasis on ensuring that quality is maintained in all portions of the development life cycle. This standard is significant for our project as it is all software, and how we develop our software will have an extreme impact on ensuring product quality.

Another vital standard to our project is IEEE 1012-1998, which entails software verification and validation information. The primary purpose of this standard is to ensure that testing is done correctly and often to catch software bugs as early as possible within the software development cycle. Verification ensures the software is being developed properly to follow the specifications, which may involve code reviews, testing, etc. On the other hand, validation confirms that the developed software meets the requirements of the client/users. Further, verification answers if the software is being built correctly, whereas validation answers if the right software is being built. Firstly, the standard outlines that the requirements must be adequately defined, then the design must be created to satisfy the requirements. The implementation occurs, including testing, code reviews, etc.; finally, the testing involves validating that the software works as intended. Importantly, given that this is an entire software project, there may be some going back and forth between the steps, but the general structure still stands. Overall, this standard intends to ensure that the product being developed is not on the right product, meaning it fulfills the users' requirements, but also that the product is being developed properly to perform the desired function correctly and within the specifications.

A third necessary standard of our project is IEEE/ISO/IEC 29119-2-2021, which directly deals with software testing, such as how to test software systematically. As mentioned in the first standard discussed, IEEE 1448a-1996, some of the test process framework reappears here with organizational, test management, and dynamic test processes. This framework helps ensure that the testing process has consistency throughout. Specifically, organizational test processes involve defining objectives, how testing will be performed, such as what methods, tools, etc., and how testing will be monitored to make necessary improvements. Test management processes involve the planning of tests, such as the general approach to testing, the resources required, and how frequently to test. It also involves tracking the progress of the various tests and reviewing test results to ensure that they are completed as intended. Lastly, dynamic test processes cover the actual execution of test design, setup, and execution. Test design involves designing tests to ensure that everything is being tested, such as all requirements being tested. Test setup involves making sure that access to all necessary hardware/software is had. Test execution includes conducting the tests manually, via a pipeline, or in some other manner. Overall, this is an essential standard for our project, as testing is how we will determine that our product meets the requirements and needs of our client.

2.2.3 Rationale for Applicability

IEEE 1448a-1996: This standard aligns well with Agile, offering a robust framework for managing the software life cycle that supports our iterative and flexible development approach. Each Agile sprint can correspond to a cycle through the life cycle phases, ensuring we consistently revisit planning, analysis, implementation, and testing. This will help us maintain quality and manage the complexities of our AWS infrastructure, including deployments in S3, CloudFront, and API Gateway.

IEEE 1012-1998: Verification and validation are essential in Agile as we iterate and add new features with each sprint. This standard ensures that frequent testing, code reviews, and requirement validation are performed consistently. We can automate much of the verification process with AWS tools like CloudWatch and Gitlab pipelines. At the same time, manual validations will help us confirm that the app's functionality aligns with client requirements at every stage.

IEEE/ISO/IEC 29119-2-2021: Systematic testing is critical for both our agile process and our AWS-based backend, which includes several interconnected services (API Gateway, Lambda, Aurora MySQL). This standard's framework provides a way for us to ensure consistency in the testing process across all AWS components. By integrating automated testing through Gitlab pipelines and comprehensive test management practices, we can validate any changes to the app and minimize risks associated with cloud deployments.

2.2.4 Other Standards

IEEE 1471-2000: This is the IEEE standard for Recommended Practice for Architectural Description for Software-Intensive Systems. A few team members brought this up because our project will include both a website and an application, so it is important that we have a strong framework so that the app and the website can properly work together and our code can support both of them.

IEEE 1008-1987: This is the IEEE standard for Software Unit Testing. While our team officially decided for IEEE 29119-2-2021 to be included in our top three standards to cover our system testing, it was also brought up. Specifically, unit testing would be important for our project. Unit testing allows us to test all of the individual components in our app and ensure that they work. All of our javascript functions and API calls would be dependent on unit testing on the micro level in addition to the full system level tests, so it is essential for there to be a consistent standard for these tests.

IEEE 829-2008: This is the IEEE standard for Software and System Test Documentation. A group member also brought up this standard, as the procedures used to test must be consistent throughout the app. This ensures that all components are tested rigorously and that no parts are left out of testing. Proper documentation is essential to clarify how and when items were tested.

2.2.5 Modifications needed to incorporate these standards

1. For standard IEEE 1448a-1996, "Standard for Information Technology - Software Life Cycle Processes," we intend to be deliberate and consistent with our processes supporting the software development life cycle. One of the biggest things we intend to do is to write detailed documentation for both backend and frontend code. This could be API endpoints' descriptions, URLs, and intended uses for the backend. We could also

write documentation for our SQL tables, with descriptions of the columns. For the front end, we could write documentation about the different screens, what API endpoints they utilize, and what information they need to display correctly. We could also write documentation for potential modals and popups within the front end to describe the conditions for them to appear.

2. We intend to incorporate systematic verification and validation within our development system to comply with standard IEEE 1012-1998. These processes can largely be automated using AWS tools like CloudWatch to monitor the deployment of our app to AWS, along with utilizing the GitLab pipeline to ensure commits are safe and runnable. We also intend to do manual validation and verification through testing and code review before each commit for extra assurance.
3. For standard IEEE/ISO/IEC 29119-2-2021, we intend to test each of our commits to ensure our code is stable, secure, and bug-free. This is most important for API integrations to ensure data is secure and inaccessible for unintended use cases. However, it is also essential that we do routine front-end testing to ensure our application is stable and won't crash or behave incorrectly. While we intend to use the GitLab pipeline to ensure our code runs without issues, we also plan to do comprehensive self-reviews and sophisticated system and integration testing before each commits to catch logic errors the pipeline may not catch. Along with this, we also plan to do peer reviews for commits with major implications and of significant importance to the app's integrity to ensure there aren't any uncaught app-breaking bugs or vulnerabilities.

3 Project Plan

3.1 Project Management and Tracking Procedures

Due to its iterative and flexible approach, the team chose Agile methodologies for project management strategy. Since the project is entirely software-based, we can leverage agile to break the project into smaller, more manageable tasks. This enables us to prioritize and deliver critical features—such as setting up the AWS infrastructure or implementing front-end post-creation—early in the process. Therefore, we can develop and refine the core functionality of the project, allowing for a continuous improvement loop based on changing requirements. Adopting Agile methodologies allows the ability to quickly adapt to feedback from the project sponsor, which is crucial for a dynamic project environment. Agile encourages a high level of continuous improvement and collaboration to make sure the team maintains alignment with the end-user's needs.
Several tools were used to ensure the team stayed on track and maintained frequent communication for this project:

- **Git/GitLab**: For version control and collaboration on code, we will utilize Git and GitLab. This will allow us to manage different branches, review code through pull requests, and maintain a clear history of all project changes. GitLab will also serve as a platform for continuous integration and deployment of new code.

- **Issue Board**: The project tasks are organized into a Kanban-styled GitLab issue board. The board will contain columns including "To Do," "In Progress," and "Completed." Each issue will be assigned to one of these groups. Each issue will have a single team member assigned to it, allowing real-time visibility and information regarding the status of various project components.
- **Discord**: The team has a discord server for all communication forms. Dedicated channels were created for various aspects of the project (e.g., “Frontend,” “Backend,” “Meetings”) to organize the discussions. Users can ping others to notify them of more urgent matters to receive faster responses.
- **Google Drive**: For document storage, we have been using a shared Google Drive folder. All shared assignments are located in this location, For organization, we have subfolders for common elements, such as a weekly report folder and a design document folder.
- **Weekly Meetings**: The team meets every Thursday afternoon from 2:00 to 3:00 PM for a standup meeting. During this time, each member explains what they worked on, what they plan to take on next, and any blockers they face. These meetings keep the entire team updated on the project’s most recent development.
- **Sprints and Retrospectives**: This team has two-week sprints. To begin each sprint, we have a planning session to identify and prioritize the tasks for the next sprint. This involves creating a product backlog and estimating each feature's time. Following each sprint, we have a sprint retrospective meeting to reflect on what went well and poorly and how we can improve for the subsequent sprint. This feedback loop is useful for optimizing the overall workflow and increasing productivity.

3.2 Task Decomposition

At the top level of our project, we have the dripdrop apparel platform. This entails multiple large tasks such as creating tables and API endpoints for our backend, a mobile application for Android and iOS, a website, infrastructure, and AI-based features. To further describe these tasks, the portion of the table is how we will store all of the user data and how the API will connect our backend to our front end. The mobile applications for Android and iOS are two main ways users can interact with our dripdrop platform. The third and final primary way of user interaction will be provided through the website, which will have the same functionality as the applications. The infrastructure will involve a CI/CD pipeline, database management, and website hosting. Finally, the AI features will involve tasks such as determining the dataset to train the model on, deciding what model to use, and determining how to train said model.
The database is created using Amazon Aurora, which will house all our essential tables with the necessary information for our platform. This includes, but is not limited to, a table for user information, posts, images for items/posts, the items themselves, coordinates for each item (i.e., where in the picture an item exists), clothing items that house general clothing item information such as price, brand, category, etc., and a tag table to store tag information for clothing items. Further, the backend involves an API requiring lambda functions to be implemented so that the front end can request information, create, delete, and more. Specifically, these lambdas include,

but are not limited to, basic CRUD operations (creation, reading, updating, and deletion). Finally, the backend must also store the images using S3 Buckets.
These will be how users interact with our social media platform for the applications and the website. For these tasks, sign-up/sign-in pages and functionality will need to be implemented: the ability to take pictures via the app and the ability to upload pictures on both the app and the website, a posts/feed page unique to each user, tailored to their likings, and the usage of AI to suggest similar products to users so one can find a similar looking outfit at a cheaper price. Further, we need to create functional notifications and wish lists/saved posts so users can revisit specific posts or clothing items later.
The infrastructure involves hosting the website, AWS CDK, database management, and a CI/CD pipeline. Website hosting is how we will host our website so that it is public and users can interact with it. As for the AWS CDK (AWS Cloud Development Kit), this defines our cloud infrastructure meaning that we can create reusable components (constructs) for easier deployment of our work. Infrastructure also involves database management, which defines how we organize and manage all tables we create for proper and efficient data storage. Finally, the infrastructure also involves a CI/CD pipeline to ensure our desired functionality continues throughout integration/development.
The AI features involve determining what dataset we will train an AI model on, what AI model we want to train in the first place, and how we want to train it. Determining the dataset, model, and how we will train the AI will involve researching various options and then considering the pricing and effectiveness of each option. This ensures our AI model performs our desired actions efficiently and effectively.
In general, we have a lot of high-importance tasks that must be done and executed properly, such as the design and implementation of the backend, the applications via React Native, the website via React, the planning and implementation of the infrastructure components, and the proper execution of developing our AI model. All of these high-level tasks involve many low-level tasks, as described above, that are all important and necessary for the success of our product.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestone 1 - Website

Our first key milestone will be to have a functioning website. This can be broken down into multiple parts, including setting up the database, the backend, the API, and having a functional frontend for users to interact.
For the front end, our first metric is a response time of <2 seconds. When a user wants to load a new page or act, we want the information to be brought over the API to the backend and then to the frontend in <2 seconds. This metric is strict, but we want the app to create a fast and efficient user experience.
On the API level, there will be many metrics that we can measure. The first is the error rate. We have a goal error rate of 1 percent or less. We want 1 percent of our backend calls to fail because, again, we want a smooth aesthetic feel for our users, and errors from our product would take away from that. We want our latency to be <200 ms for data transfers on the

website. This ties in with keeping response time at <1 second, but specifically for latency, we do not want more than 200 ms before data transfers can begin. We also want the throughput of calls to be >30 RPS(responses per second). This value will be fast enough for our app to perform with a moderate amount of users without issues. When our app grows, we will have to increase the throughput, but for our prediction of a moderate amount of users at the start of the app, 30 RPS will suffice.
For the backend we want the backend to be able to support 100 concurrent users. This concurrency rate will support our product when it is moderately sized and will have to be optimized if our app grows to be larger, but for the initial release of the website, this is our goal. Regarding our database queries, the metric is that simple queries are completed in < 200ms and complex queries are completed in <500ms. This will allow the website to quickly post data and get information from the database and keep the overall request quick.

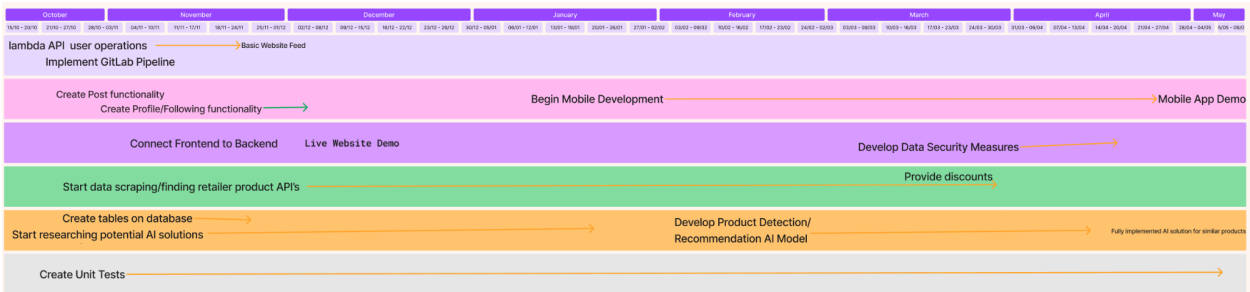
Milestone 2- Mobile Application

Our next key milestone is the release of the app. This app will have the same backend as the website, so many metrics will be very similar. In terms of producing the app, we want the app to work on both IOS and Android products. The app will also be able to send mobile notifications. Our performance metrics for the API, backend, and database will be the same for the app as they were for the website because we will use the same backend for both. One metric we wanted to add for the app is that the app fully loads in <4 seconds for the user to interact with.

Milestone 3 - AI to Find Similar Product

Our final key milestone is releasing the AI-generated list of similar products that users can click on when looking at a post. This is the most complex portion of the entire product, as we are creating our own AI model to do this. For metrics for the AI model, we expect the matched products to match 85% of the tags of the original apparel item. One of the functionalities of our AI is that it will generate tags (such as jeans, light blue, faded, size 28x30, bellbottom, etc.) This 85% match rate means that for an item to be placed on the view similar items list, it must match 85% of these tags. We want the error rate of this list to be less than %5, so less than 1 in 20 products on the list should not belong there (not meet the 85% criteria).

3.4 Project Timeline/Schedule



As described above, our project has three major milestones, each with several subtasks and timelines. The first milestone we want is to have a working website. We hope to have a version ready to demo by the end of the first semester in mid-December. As more features are considered, we will continue expanding our website throughout the second semester. Before we have a working website, we need to develop an API using lambda based on a database that will be mostly complete by the beginning of December. We plan to complete most of our lambda functions by late November. We will continuously update our front end to incorporate our API calls while they're developed. To have good testing measures, we developed a GitLab pipeline throughout October to track deployment progress and check for bugs. We will also continue to create unit tests throughout the project's lifespan to ensure as few bugs as possible. We also plan to develop code to scrape several sites routinely to find good deals for articles of clothing. We anticipate this will be a large-scale project and last until early April. Near the end of this process, we will provide these discounts to our users once we have enough sources to do so at an acceptable level. We also plan to develop measures to keep data secure throughout the two-semester project, especially near March, once we have enough data to warrant it.
Our second major milestone is a mobile application. The mobile app development will start in mid-January and last until the beginning of May. In mid-May, we will demo our mobile app. Most of our app's front-end screens will take significant inspiration from the website, so we anticipate the app to be at the same level of completeness as the website by early March. After that, we will simultaneously continue to develop both platforms.
The third major milestone is building a custom AI model for detecting and recommending clothing items. This will likely be our most challenging milestone. We have started researching methods and ideas for our model and will continue to learn about AI and develop our plan through mid-January. Once we have a strong understanding of our plan, we will start developing our model, which will last through the end of the semester in mid-May.

3.5 Risks and Risk Management/Mitigation

3.5.1 Frontend

1. User Authentication

Risk: Our authentication process could have security vulnerabilities
Probability: 0.2

2. Creating Posts

Risk: Users uploading inappropriate text or images
Probability: 0.5
Mitigation: Complete thorough research on ways to implement filtering + perform thorough testing to ensure consistent accuracy of filtering

3. Scrollable Feed

Risk: Feed could be slow or buggy when we reach a high volume of posts
Probability: 0.6
Mitigation: Implement pagination or infinite scroll + use caching for recently viewed posts

4. User Profile

Risk: User profile screen and features may not be user-friendly
Probability: 0.2

5. Saved Posts

Risk: This feature may load slowly if we have to query all the saved posts for a user every time we load this page
Probability: 0.5
Mitigation: Use efficient indexing in the database; consider limits on saved posts or archiving strategies.

6. Price Comparison Feature

Risk: This is a very complex feature that relies on AI. As a result, the user may experience a buggy or slow experience.
Probability: 0.8
Mitigation: Identify ways to improve the app's usability if we experience unexpected responses or behavior from the AI. Handle long response times with loading visuals.

3.5.2 Backend

1. Creating Tables

Risk: Complex relationships that cause errors and require many API calls on the frontend
Probability: 0.5
Mitigation: Be thorough with our planning phase of the tables, and consult the team when making changes or updates to our original design

2. API Endpoints

Risk: Certain endpoints could experience high latency or traffic
Probability: 0.3

3. Lambda Functions

Risk: Lambdas breaking after new changes will cause our app to break
Probability: 0.6
Mitigation: Implement unit tests that must pass before new changes are pushed to the master branch

4. Storing Images

Risk: Running out of space to store all the images
Probability: 0.2

5. Research 3rd Party API for retail data

Risk: Spend too much time looking for a perfect solution instead of creating our own
Probability: 0.7
Mitigation: We need to do this research early on so that we have time to plan or switch directions depending on what we choose

3.5.3 Infrastructure

1. Hosting Website

Risk: Unexpected downtime + cost increases
Probability: 0.1

2. CDK code

Risk: Developers make changes on the AWS console and cause issues to our infrastructure
Probability: 0.5
Mitigation: Make it clear to all teammates that all infrastructure changes need to be made using our CDK code.

3. Database Management

Risk: Incorrect configuration or provisioning could lead to overuse and overspending of resources.
Probability: 0.6
Mitigation: Closely monitor our AWS bill and budget and make changes if we notice anything unusual. Also, set an AWS budget to monitor us when we are close to our monthly budget

4. Pipeline

Risk: Automated CI/CD could deploy unwanted or untested code.
Probability: 0.2

3.5.4 Artificial Intelligence

1. Determine Dataset

Risk: Finding a high-quality dataset with the necessary diversity and accuracy could be challenging and time-consuming
Probability: 0.8
Mitigation: Start this process early and find alternate solutions to fall back on.

2. Determine the Model to Use

Risk: There are many models out there, and finding the one that works best for the context of this class may not be feasible
Probability: 0.7
Mitigation: Similar to the previous risk, we need to progress on this early so we can pivot/change if necessary.

3. Train the Model

Risk: Training infrastructure could exceed budget or fail to handle large data.
Probability: 0.6
Mitigation: Train using cloud GPU/TPU resources on a pay-as-you-go basis; if budgetary constraints arise, consider transfer learning or using a pre-trained model to reduce time and cost.

3.6 Personnel Effort Requirements

Total Hours Worked	
Tasks	**Estimated Hours**
Frontend	
User Authentication	10
Creating Posts	5
Scrollable Feed	5
User Profile	15
Saved Posts	10
Price Comparison	20
Backend	
Creating the Tables	5
Creating API endpoints	5
Writing the lambda functions	25
Storing images	10
Researching 3rd party APIs	15
Infrastructure	
Hosting Website	5
CDK code	30
Database Managment	5
Pipeline	10
	175

This table provides a structured overview of tasks along with estimated hours for each task in the "Frontend," "Backend," and "Infrastructure" sections. The columns to the right of the estimated hours are designated for group members to log their time spent on each task. This will help track actual hours against the estimated hours and assist in monitoring individual contributions and task progress.

3.7 Other Resource Requirements

Our main resource requirement is our AWS account. We have contacted ETG and received an AWS account we all can access. We also sent them a monthly budget projection and had that approved. Currently, AWS is the only resource we are using, because we are taking advantage of their cloud computing features for all aspects of our project that require computing resources. In the future, depending on what AI solution we determine is best for our product matching feature, we will need to add an AI resource to this section.

4. Design Exploration

4.1 Design Decisions

When planning the architecture, the team prioritized scalability, flexibility, and cost-effectiveness, which led us to leverage the cloud for the infrastructure, specifically AWS, as the cloud provider of choice. Furthermore, when selecting a database, Amazon Aurora, a MySQL database, was selected due to its high speed and scalability compared to alternatives. Once this was

determined, the frontend language needed to be determined. Due to its flexibility as a javascript library focusing on the view layer, the team decided to leverage React.

4.1.1 AWS vs. Azure

Using a cloud provider over on-premise is crucial for a more scalable, flexible, and cost-effective solution. Once we decided to use a cloud provider, we needed to determine which provider best suited our needs, and we chose to use AWS over Microsoft Azure for several reasons. First, our decision to use AI for product recommendations is a key feature of our app, and AWS offers more flexible and advanced services in the AI/ML realm. Specifically, Amazon SageMaker will allow us to quickly build and train an effective model to analyze clothing items and recommend similar products. Additionally, Amazon's Aurora offers a leg up on Azure SQL since we use a relational database. Aurora has superior speed and scalability while being compatible with MySQL. Azure SQL would have been the better choice if we needed a seamless integration with other Microsoft products. Still, it cannot match the low-latency reads and other benefits of Amazon Aurora.

4.1.2 AWS Aurora vs. MongoDB

After deciding which cloud provider to use, we needed to determine what kind of database would best fit the architecture of dripdrop because selecting the correct database impacts our ability to scale and maintain a positive user experience. Leveraging a database through AWS was preferred because it offers a seamless integration with other AWS services. Next, when deciding whether to leverage a relational or nonrelational database, we considered the use cases of what dripdrop will need to do. A relational database made the most sense since the app's functions require structured tables for our users, posts, clothing items, and favorite items. Aurora is a managed, relational database offering automated backups and compatibility with MySQL and PostgreSQL. On the other hand, MongoDB is a key-value NoSQL database, which is better for high-volume data ingestion and real-time data analytics. MongoDB's lack of enforced schemas would require additional validation in the application layer to ensure data quality, creating more work than having a structured database.

4.1.3 React vs. Angular

For our fronted development, we found that React would best suit the needs of our application. First, since React is a javascript library focusing only on the view layer, it allows for easier integration with other libraries and third-party tools, giving us more flexibility. Angular is an MVC framework that is helpful for end-to-end solutions but can limit flexibility and make it more challenging to integrate with other tools and libraries if we need to, down the road. Being open source, React also has a larger and more active developer community, including comprehensive documentation, making it effortless to follow best practices. Although Angular also has a strong community, the presence of React's community allows us to stay more agile and respond to issues faster. Lastly, our team has more experience working with React which will help us maximize our development time, ensuring project deadlines are met.

4.2 Ideation

Decision: Infrastructure/Server Provider

Early on, one of the most significant decisions we needed to make was how we would host critical aspects of our project, like our database and front-end code. As discussed above, the primary decision was Microsoft Azure vs. AWS.
Initially, we identified 3 potential paths to take: using on-premise Iowa State servers, a cloud provider, or combining the two to employ a hybrid approach. There wasn't much time spent exploring more options because for servers and compute services, we have to use what Iowa State will provide us or find suitable cloud options. There isn't another feasible alternative within the scope of this class. After researching cloud providers, we decided it would be easiest to choose one of the three industry leaders: AWS, Microsoft Azure, or Google Cloud Platform. At this point, we had five potential options:

Iowa State Servers: This would require collaboration with the computer science or computer engineering department to provision servers for us. The main benefit is that all team members are familiar with this setup, and it could offer better control over data security and compliance.
Amazon Web Services: AWS is the most popular and largest cloud provider, offering many services, including computing, storage, databases, machine learning, and more. AWS is known for its scalability, security, and flexibility.
Microsoft Azure: Azure is strong for organizations with existing Microsoft infrastructure. It offers a wide range of cloud services. It is known for its hybrid capabilities, making it a good choice for enterprises looking to integrate on-premises infrastructure with the cloud.
Google Cloud Platform: GCP has strengths in data analytics, machine learning, and Kubernetes-based deployments. It also has a wide range of cloud services similar to AWS and Azure, although it is not as popular or advanced as Azure or AWS.
Hybrid Approach: This approach would combine the Iowa State servers with a cloud provider. For example, we could host non-critical data on campus servers and run high workloads in the cloud, giving us flexibility and control.

4.3 Decision-Making and Trade-Off

With 5 options to choose from, it was essential to identify essential criteria that we could use to compare the different options. We brainstormed what things were most important for our app and our team. We narrowed it down to 7 criteria to base our decision: # of team members with experience, cost, database capabilities, services provided, AI/ML capabilities, scalability, and ease of use.
After researching the five options and discussing them as a group, we ranked them for each criterion. For each row, we gave the best option a 5, the second best option a 4 … and the worst option a 1.

	Iowa State Resources	AWS	Azure	GCP	Hybrid
Team experience	5	3	2	1	4
Cost	5	3	1	2	4
Database	1	5	4	3	2
Range of Services	1	5	4	3	2
AI/ML	1	5	3	4	2
Scalability	1	5	3	4	2
Ease of Use	2	5	3	4	1
Total	16	31	20	20	17

From this table, you can see that our highest score was AWS. This is what we ended up choosing for our infrastructure and server provider. This is a result of several key reasons highlighted in the table:

- Of all the cloud options, our team had the most prior experience working with AWS, giving us confidence we would be able to implement this choice effectively
- Looking at cost, AWS again scored the best of the cloud options. The AWS free tier offers many ways to take advantage of their popular services for free. Although it may be more expensive than purely Iowa State resources, its benefits outweigh these costs.
- When you dive into the services provided by each option, AWS is again the leader. AWS didn't become the most used cloud service by accident; it has the most robust and wide variety of services on the market. If we used Iowa State resources, we would have to use many individual third-party services, whereas AWS has a service for just about everything.
- While a hybrid approach could take advantage of many of these AWS benefits, our group believed the challenges and complexity of managing AWS and Iowa State resources simultaneously make it the hardest approach to implement.

4.4 Proposed Design

4.4.1 Overview

Our app, dripdrop, is a social media apparel posting app in which users post their favorite outfits and provide the URL where they bought the items. Our app will then use this URL to generate the product information for the viewers of the post to see when they click on the post. The idea

is that our app, dripdrop, will partner with many apparel companies. Then, when users post on our app and other people use their posts to find and buy items, our company will receive a commission for the purchased items, and we will trickle most of this commission down to the user that posted, but in the form of dripdrop points. The points can be cashed in for gift cards to all our partner companies. So, we are allowing users to play the role of an influencer without first building a following or creating their own partnerships with apparel companies, which can be a lot of work.

To provide a clear visual, here is how a user can tap on a post to view the tagged items:

View **Tap** **Shop**

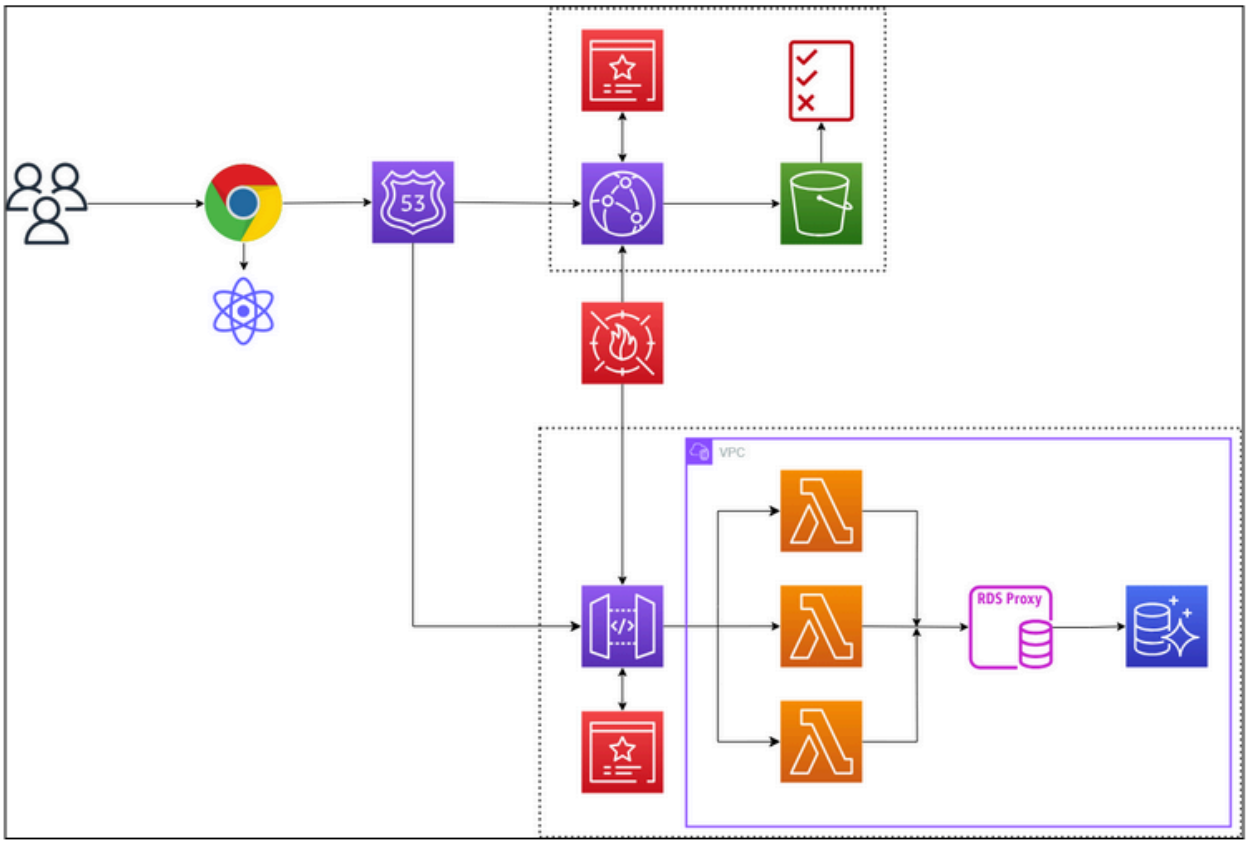
It is as simple as viewing the item, tapping on the post to see which items are tagged, and then tapping/hovering over the desired item's tags, which creates a pop-up of the item info and link.

There are going to be several features that go with this design. The first and largest will be our "find similar items" button attached to the apparel tags on every post.. This will be an AI generated list of items that look similar to the original item but are cheaper so that the user can get the looks of the outfit without spending as much.

Another feature will be the ability to follow other users and have a following. This will be similar to what other social media platforms do, so the people you follow will be the primary people who will pop up on the feed of posts.

Users will also be able to save posts and individual apparel items. These will be stored on their profile page and allow users to go back to old posts they looked at to see if products have gone on sale or are simply in a better place to buy. Additionally, we will have a search bar so user profiles, posts, and specific items in posts can be searched using the search bar.

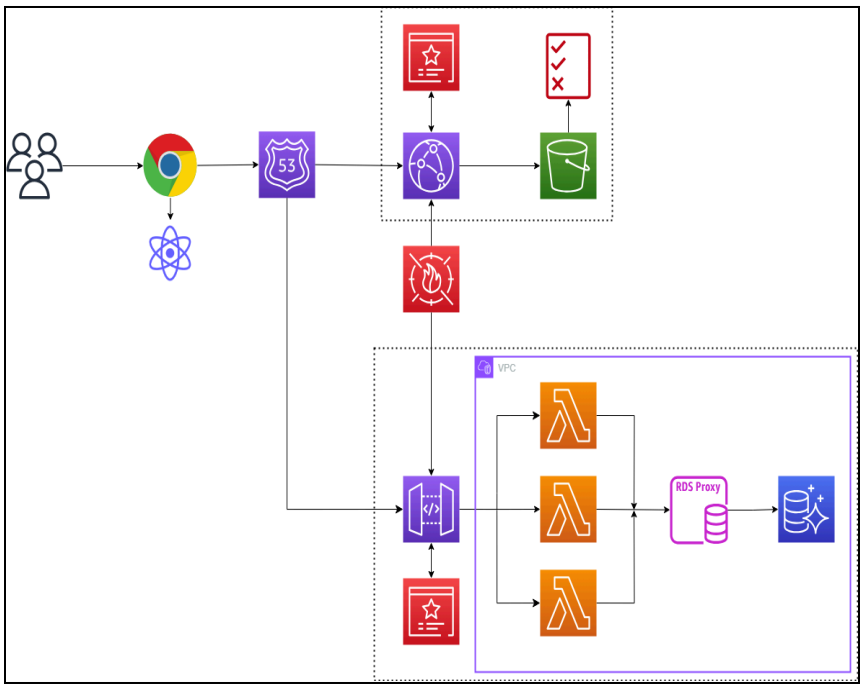
4.4.2 Detailed Design and Visual(s)



4.4.2.1 High-Level Overview

The system design comprises several AWS services that provide a scalable, secure, and performant application infrastructure. The design is broken down into subsystems, each playing a distinct role, from user interface (UI) to backend processing and data storage. Key components include CloudFront for content delivery, Route 53 for DNS management, an S3 bucket for static content, API Gateway for managing API requests, Lambda functions for business logic, and Aurora MySQL for data persistence. This architecture follows a serverless approach where services are highly managed, reducing the operational overhead and ensuring scalability.

4.4.2.2 Detailed Overview (API)



This API infrastructure is designed with AWS Lambda functions as the core of its serverless backend, each handling specific API operations. The API leverages API Gateway to manage HTTP requests, which trigger the appropriate Lambda function based on the request path and method. The Aurora MySQL database serves as the primary data storage, and it is connected via RDS Proxy to manage connection pooling and optimize performance under load. AWS Secrets Manager securely stores database credentials, and each component operates within a Virtual Private Cloud (VPC) to enhance security.

4.4.2.2.1 Key Components and Their Roles

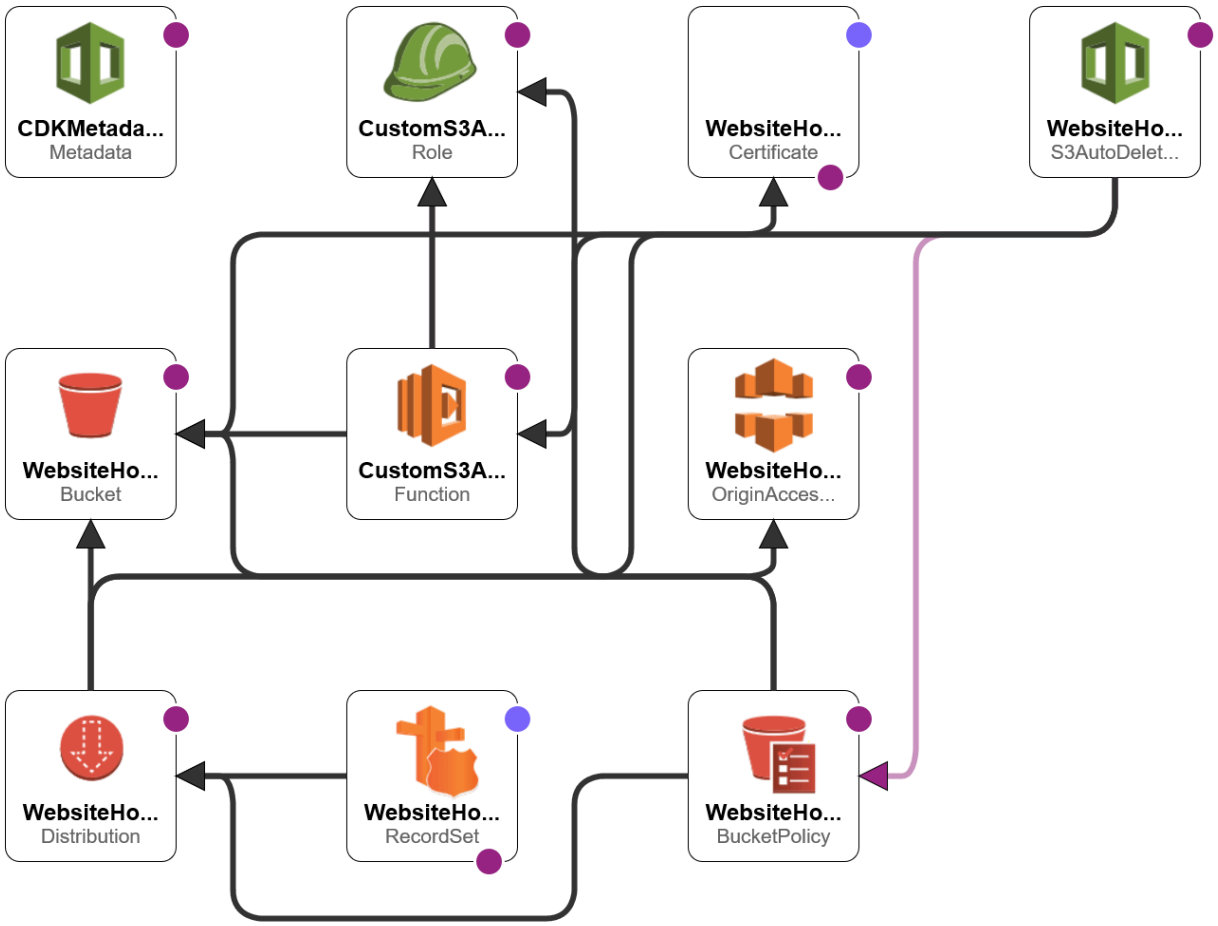
1. **API Gateway**: Acts as the entry point for all HTTP requests to the API, routing them to the appropriate Lambda function based on the request's path and method. It also enforces security and access control policies.
2. **Lambda Functions**:
    - Each Lambda function serves a specific or a set of related endpoints for CRUD operations.
    - Functions include CreateUser, GetUsers, GetUserById, UpdateUser, DeleteUser, and authentication (UserSignIn). Similar Lambda functions manage Post and Image data.
    - Each function operates in a VPC, has access to Secrets Manager for database credentials, and uses the shared environment configuration to interact with the Aurora MySQL database via RDS Proxy.
3. **Aurora MySQL Database**:
    - Acts as the persistent storage for user, post, and image data.
    - Connected to Lambda functions through RDS Proxy helps manage database connections efficiently, especially in high-throughput scenarios.

4. **RDS Proxy**:
    - Provides connection pooling and efficient database connection management for Lambda functions.
    - Protects the database from being overwhelmed by managing simultaneous connections from multiple Lambda functions, thus enhancing reliability and scalability.
5. **Secrets Manager**:
    - Securely stores database credentials, which Lambda functions retrieve at runtime to establish secure connections to Aurora MySQL.
    - Integrated with IAM roles and policies, only specific Lambda functions can retrieve the database credentials.
6. **VPC and Networking**:
    - A VPC isolates the API components to ensure security and controlled access.
    - Private subnets for Lambda functions and Aurora MySQL prevent direct internet exposure.
    - Security groups restrict traffic to and from Lambda functions and the Aurora database, ensuring that only permitted connections are allowed

4.4.2.2.2 Internal Operations and Flow

1. **User Requests**: Users interact with the API via HTTP requests sent to the API Gateway.
    - API Gateway determines the correct Lambda function to invoke based on the URL path and HTTP method.
    - For instance, a POST request to /users would trigger the CreateUserLambda, while a GET request to /users/{id} would trigger GetUserByIdLambda.
2. **Lambda Function Execution**:
    - Each Lambda function retrieves necessary configuration data (e.g., database endpoint, port, credentials) from its environment variables and Secrets Manager.
    - The function executes the corresponding business logic (e.g., creating, retrieving, updating, or deleting a user).
    - It uses RDS Proxy to establish a connection to the Aurora database, ensuring efficient use of database connections and reducing latency.
3. **Database Operations**:
    - Lambda functions interact with the Aurora MySQL database for all data persistence needs.
    - RDS Proxy manages database connection pooling, minimizing overhead and handling failover in case of connection issues.

4.4.2.3 Detailed Overview (Static Website Hosting)



The architecture is designed to host a static website on AWS using Amazon S3, CloudFront, and Route 53. The stack includes components for content delivery, security, and automated bucket management. Amazon S3 stores static content, CloudFront is the content delivery network (CDN) for fast and secure access, and Route 53 provides DNS management for the custom domain.

4.4.2.3.1 Key Components and Their Roles

1. **Amazon S3**:
    - Role: The primary storage for the website's static assets (HTML, CSS, JavaScript, images, etc.).
    - Configuration:
        - The S3 bucket is configured to block all public access, ensuring the content is only accessible via CloudFront.
        - Objects are auto-deleted through a custom Lambda function to manage and clean up objects when necessary automatically.
    - Bucket Policy: Configures permissions to allow CloudFront to retrieve objects from the bucket securely.
2. **CloudFront:**

- Role: Provides a globally distributed content delivery network to ensure low-latency access to the website's static content.
- Configuration:
  - Uses an SSL certificate for secure HTTPS connections, enforcing TLSv1.2 and above.
  - Configures custom error responses, such as a 403 response directing to error.html.
  - Caches content for improved performance and minimizes requests to the S3 bucket.
  - Restricts access to the S3 bucket via Origin Access Control (OAC), preventing direct public access to S3.
  - The ViewerProtocolPolicy is set to redirect to HTTPS to enforce secure connections.

3. **Route 53**:
   - Role: Manages DNS records to route traffic to the CloudFront distribution.
   - Configuration:
     - An alias record links the custom domain (www.dripdropco.com) to the CloudFront distribution, enabling access via the custom URL.
     - Uses hosted zone information to configure the DNS correctly for CloudFront.
4. **AWS Certificate Manager**:
   - Role: Manages the SSL/TLS certificate for secure HTTPS connections.
   - Configuration:
     - The certificate is validated using DNS with Route 53.
     - Applied to the CloudFront distribution to enable SSL/TLS encryption.
5. **Lambda Function for Auto-Delete**:
   - Role: A custom resource for automatically deleting objects within the S3 bucket.
   - Configuration:
     - A Lambda function, triggered by AWS custom events, deletes objects in the S3 bucket as needed, helping with storage management.
     - Runs with the AWSLambdaBasicExecutionRole for necessary permissions.

4.4.2.3.2 Internal Operations and Flow

1. **User Access**:
   - Users access the website via www.dripdropco.com.
   - Route 53 directs traffic to the CloudFront distribution, which serves the static content stored in the S3 bucket.
2. **Content Delivery**:
   - CloudFront fetches content from the S3 bucket when not cached and serves it from edge locations, reducing latency and improving load times for global users.
   - Only CloudFront has permission to access the S3 bucket, controlled via OAC and bucket policies, ensuring secure access to website resources.
3. **Error Handling**:

    - Custom error responses in CloudFront direct users to error.html in case of a 403 (Forbidden) error, enhancing the user experience.
4. **SSL/TLS Security**:
    - SSL/TLS is enforced by CloudFront using the certificate issued by AWS Certificate Manager, ensuring encrypted and secure connections.
5. **Automated Cleanup**:
    - The Lambda function automatically deletes objects within the S3 bucket as specified, managing storage and ensuring the bucket remains clean.

4.4.2.3.3 Security and Access Control

- **IAM Roles and Policies**: Each Lambda function is associated with an IAM role that grants it access to the necessary resources, including permissions to retrieve secrets from Secrets Manager and access the database through RDS Proxy.
- **Security Groups**: Lambda functions and the Aurora MySQL database are assigned security groups that define the inbound and outbound traffic rules, restricting access to only what's necessary for each component.
- **Bucket Access Control**: The S3 bucket blocks all public access, and only CloudFront can access it via Origin Access Control.
- **CloudFront Security**: The CloudFront distribution is configured to enforce HTTPS, redirecting all HTTP requests to HTTPS and ensuring secure connections.

4.4.3 Functionality

Overview: dripdrop is meant to be an easy-to-use, understand, and operate social media platform that allows users to have a singular place to find outfits they like, cheaper options for a similar look, and the ability to earn points for sharing their outfits with others. To achieve these goals, we have designed the UI to be easy to understand and navigate for any user, as it is intuitive and easy on the eyes. We have also decided on functionality to provide users with solutions to these desires.
UI: For the UI, we have made original sketches to get an idea of what our UI should look like so that we can be consistent and all come to a consensus on major design decisions. By doing so, we can create an intuitive and straightforward design that all users can easily understand and use. For example, if a user wants to find a specific function or section of the application/website, the navigation naming is straightforward so it should be easy to find.
Feed: For users to find outfits they like, we will have a feed, the first page shown upon entering the website or the application, displaying popular outfits for the user to scroll through and find something they like. Users can also follow other users if they have a style they like, and they can save specific posts if they want to return to those outfits later.
Product Suggestions: We have used AI to implement said functionality to find cheaper options for a similar look. Using AI, we can tag various products and then use those tags to find similar products to the current product, at cheaper prices. If a user were to navigate to an apparel item that they like, but the price is too high, they can simply look below the product listing and see the suggested similar products section. This section will have similar apparel items so the user

can quickly scroll through these similar products and find something else they like at a cheaper, more affordable price.
Reward System: Rewarding the users who post outfits is an important aspect of our social media platform. That is how we attract users and how they can make money from simply sharing their outfits with everyone else, regardless of their following. A user simply has to post their outfit and provide the links to the products, and we can use our partnerships to earn money from other users purchasing said outfit from the provided links. This results in us earning commissions and the poster earning dripdrop points which can be used towards purchasing apparel items.

4.3.4 Areas of Concern and Development

Our current design satisfies many of the user's primary requirements and needs. Some of the requirements we are confident about satisfying include the ability for users to sign up/login for accounts and to view and create posts for other users to see in the feed. Additionally, we will integrate the “following” feature within the next couple of weeks to allow users to follow each other and prioritize their following list on their post feed. Further, we have added profile pages where users can view each others’ profiles and posts. We will also introduce user and profile settings over the next few weeks, including changing the username and password, whether a profile is public or private, and other basic settings.
While we feel strongly that we will have many of our users’ needs satisfied by the end of the semester, there are some areas of concern that we think will take longer to resolve. One of the most significant issues we’re facing is the ability to tag individual articles of clothing when making a post, discovering new items, and finding deals for those items. This process requires gathering information from the brand’s website, which is difficult because there are a lot of different brands of clothes, each with varying formats of website, making it hard to automate the data collection process through practices like scraping. A solution we think will help to alleviate these difficulties is to start by offering a handful of more prominent brands to be tagged, automating the data collection process for those websites, and gradually expanding our offering rather than having an extensive collection all at once. We believe that we will be able to accommodate at least ten different brands along with their catalogs by the end of the school year.
Another area of concern that we have is regarding our AI integration. Our design promotes a custom-built AI model that will be able to identify specific articles of clothing and their corresponding brands within a post to help simplify the tagging process. However, due to the vast number of brands and types of clothes, it will be difficult to train our model to be sufficiently accurate by the end of the school year. One of the solutions we discussed was to simplify our AI model to identify essential features of individual articles of clothing (like “black t-shirt” or “ripped light blue jeans”) rather than having the model guess the brand. From there we could create a script to correlate these tags to an item in our database with similar tags and order by popularity to give a reasonable estimate of the item.
Our final primary concern concerns our concept of “dripdrop points” earned through affiliate links. While we feel this could be a relatively easy integration, we need to research what companies offer referral codes and the logistics of using referral codes. Additionally, we need to

discuss further the primary motivation for users to use the app and find a good balance of offering posting incentives while preventing over-monetization and spam. We also feel that this feature won't be released later once we have an established user base, many brands, and potential referral codes to work with. We would appreciate it if our TAs or faculty advisers have any insights or opinions on ways to incorporate a referral system.

4.4 Technology Considerations

Overview: We had some critical technology decisions to make since this would decide the entire structure of our social media platform. This could and would directly impact possibilities, costs, and opportunities as certain options would excel in areas that others would not. We first decided to use AWS as our cloud provider, AWS Aurora for our database, and React for our front end.

AWS: We decided on AWS as our cloud provider due to its flexibility and scalability. Some of the most important strengths AWS has that are crucial to our project are the AI capabilities we need for our product recommendations, the speed and scalability of AWS if we need scalability in the future, and the platform's reliability is crucial. However, AWS does have more complex pricing than alternatives and generally has a steeper learning curve. The pricing is not a massive issue as the benefits that AWS provides far outweigh the cons of pricing. As for the steeper learning curve, this only proves to be an issue initially; however, using AWS's extensive documentation, we can get past this issue, and it will no longer be an issue.

AWS Aurora: For our database, we decided on AWS Aurora over other options, such as MongoDB, due to the performance and scalability of AWS Aurora and how seamlessly it integrates with AWS, which is what we previously chose for our cloud provider. Further, the security and chance of database corruption is much better on AWS Aurora than on alternatives. However, AWS Aurora has weaknesses in that document storage is not very flexible, the cost can be quite high, and it sort of locks us into using AWS as our cloud service. The document storage issue can easily be solved by using S3 Buckets for images; no other aspect of our design will cause issues here. For the cost, this is a hit we are willing to take, as, similar to the cloud service, the scalability and performance of AWS Aurora over alternatives is worth the higher cost. Finally, for it locking us into AWS as our cloud provider, that should not be an issue as we do not foresee needing to change.

React: For the front-end development, we chose React over alternatives such as Angular. The reason we chose React is because it is component-based which leads to more modularity and reusability, the fast and efficient UI updates, the flexibility of React, and the large community and extensive documentation for reference if we face issues. On the other hand, React does have its downsides in that it relies on a third-party library, given React is a JS library and can need other libraries for various functionality, and inconsistencies can arise due to a less structured approach. Library dependency should not be much of an issue as this can be a benefit in that we can pick and choose the tools we need without incorporating unnecessary dependencies. Further, we have more flexibility regarding our tools, which can lead to more possibilities. Finally, React being less structured should not lead to inconsistencies for us as we have planned this out ahead of time and are sure to update the team with design choices continuously.

4.5 Design Analysis

We have made solid progress in creating our app's overall framework. We currently have a working AWS-hosted server that supports our app and have implemented a good portion of the back and front end.

We have created some tables we will need for the app's backend. More tables will be created, but the Users, Posts, Images, and Followers tables work. We have working lambdas for all these tables, including GET, POST, DELETE, and PUT requests. These lambdas have been tested on the AWS console or using Postman, and they all work correctly.

For our API, we have a few API endpoints called, including one for getting posts to put into the feed and another for generating the profile page for a user. We plan to implement the rest of the API structure soon.

On the front end, we have pages for signing in and signing up. These are currently functional and prompt users for specific sign-in information to let them sign in. We also have a home page, a search bar, a create posts page, and a user profile page. These are all basic pages right now and do not yet contain the data we would want in our end result, but they have the basic design and outline created in React.
So far, we have only run into a few errors. The testing has passed for all of our requests, but we have a lot of testing left to do. We did have an issue with the backend getting out of sync between the console and the files and this created errors when we were testing lambdas, but this has been resolved.

Our backend was convoluted, so we plan to restructure it more efficiently. Instead of all backend requests for each table being in different files, we want to have one handler for each table, and the handler will be able to determine the type of request and then call the specific code for that request. Other than that, the plan is to move forward as planned, continue fleshing out the front end and the API more, and then diagnose issues with file structuring or implementation methods as we come across them.

5 Testing

Testing is essential to most projects involving a circuit, a process, a power system, or software.

The testing plan should connect the requirements and the design to the adopted test strategy and instruments. In this overarching introduction, give an overview of the testing strategy and your team’s overall testing philosophy. Emphasize any unique challenges to testing for your system/design.

5.1 Unit Testing

Our unit testing can be broken into three categories: backend, frontend, and API. We have not implemented any testing yet, but once we do, we plan to insert the unit tests into our pipeline to ensure that new changes don't break existing functionality.

Backend Testing

The backend units involve all the database functions for each table and all the business logic functions we implement. We will use standard unit testing by testing each function individually with controlled/mocked inputs. It will be essential to test edge cases and invalid input. We will be using Pytest, a testing framework for Python, to write these tests.

Frontend Testing

The frontend units contain all user interface components (buttons, forms, navigation, etc). To test these units, we will include component-level testing and tests that simulate user interaction. The tool we will use for this section is Jest, a react testing library.

API Testing

The units for our API consist of our API endpoints. We will send mock requests to these endpoints to test these units and verify the response. Pytest includes a request module for automating API testing and validating responses.

5.2 Interface Testing

Interface Overview: The project's four primary interfaces are the React-based Frontend, API Gateway, lambda functions, and the database. The first component, the frontend, uses React as its framework and is how users interact with the platform. Secondly, the API Gateway will allow for communication between the front and back end. For example, when a user follows someone,then a request will be made to the backend via the API Gateway to store that follow in the database, and the same for all other user-based actions and all CRUD operations. Next, the lambda functions perform the backend functionality, which is what the API Gateway "calls" to perform the request. These functions directly interact with the database. Finally, the AWS Aurora database is how all of the platform's information is stored, including, but not limited to, user accounts, post information, images, and clothing items.

Testing Unit Composition: Three primary compositions exist to test all of the interfaces properly. Firstly, the communication between the front (React) and the API Gateway can be tested by ensuring that the appropriate API request is sent when a user interacts with the platform and that said response is handled properly. This is tested, in part, by Bruno to ensure API response behavior and, in part, by frontend testing to ensure the correct API calls are triggered.

Beyond this first composition, the connection between API Gateway and the lambda functions must be tested such that the API triggers the correct lambda functions and the correct inputs and outputs are given/received. Once again, Bruno can be used for testing to disregard the front

end to narrow the testing scope, and AWS Console can be used to test API calls manually and even lambda functions.

Finally, the third composition is between the lambda functions and the database. This must be adequately tested to ensure that CRUD operations are happening as expected and are correctly stored in the database. To test this, Bruno can be used again to test at the API level, AWS Console for testing lambda functions more directly, and MySQL Workbench to ensure the database saw the desired changes.

5.3 Integration Testing

The project has two primary critical integration paths:

- React ↔ API Gateway ↔ Lambda Functions ↔ Database
    - Criticality: This path describes the core functionality of the social media platform involving all CRUD operations the user can perform on the platform, except any image based CRUD operations. If this path does not work as intended, most of the platform will not work aside from the visual aspect.
    - Testing:
        - React Testing Library: Confirms that the front end sends the correct API requests and handles any responses received.
        - Bruno: Ensure API request and response behavior is working as expected.
        - AWS Console: This may be used to perform more direct tests for the API and lambda functionality.
        - MySQL Workbench: Used to ensure that the database was altered as expected.
- Database ↔ S3 Bucket
    - Criticality: When combined with the previous path, this path describes the final step to allow CRUD operations for images. This path must work as intended to store images, one of the most critical aspects of the platform.
    - Testing:
        - AWS Console: Confirms that S3 bucket operations are being performed by lambda functions correctly.
        - MySQL Workbench: Used to ensure that the database is appropriately referencing the correct S3 objects.
        - S3 in AWS: May be used to confirm that images are being correctly uploaded to the S3 Bucket upon request.

5.4 System Testing

Unit Tests

For unit tests, the primary focus is to test individual components and isolate issues. To do this, there are a few key tests, such as testing the UI elements and all frontend functionality using React testing libraries such as Jest. Beyond the frontend testing, the backend will also need to be tested using similar testing strategies to ensure the logic works as intended, the database can be edited appropriately and read, and S3 functionality. Most backend unit testing can be done via a tool called Pytest for testing python file functionality. Having confirmed frontend and backend functionality, the connection API needs to be tested in an isolated manner, using tools such as Pytest to create mock endpoints and properly isolate the API functionality for testing.

Interface Tests

Interface tests ensure functionality between two components, such as between the React frontend and the API Gateway, between the API Gateway and lambda functions, and between the AWS Aurora database. The key tests here ensure the database can be properly accessed and edited, as needed, and that React sends the proper requests and adequately handles the responses. Tools such as Bruno, AWS Console, and MySQL Workbench may be used for this.

Integration Tests

The primary focus of the integration testing is to ensure that the entire system works from one end to the other, so from the React frontend all the way to the database and back. Ensuring functionality here is crucial to ensuring the overall functionality of the platform. Helpful tools for this sort of testing include Bruno, MySQL Workbench, and React testing libraries.

System Tests

At the system level of testing, the focus is on ensuring the entire platform works, not only for functional and non-functional requirements. The primary testing here is for scalability, to ensure high traffic can be adequately handled, performance to ensure fast and reliable functionality, and reliability to ensure that it does not break often. Though the non-functional requirements listed are highly important, the functional requirements must also be tested at this stage. Tools such as AWS CloudWatch and load testing tools such as Apache JMeter could be used to perform this crucial testing.

5.5 Regression Testing

Preventing Regression: To ensure that new additions do not break old functionality, general testing guidelines have been put in place so that when something new is added, all old functionality that could be affected by this change must either be automatically or manually tested. This ensures that old functionality does not see any issues due to new functionality being implemented.

Critical Features: The primary critical features that must not break under any circumstance are the ability to connect the frontend to the backend for all CRUD operations, which involves ensuring the functionality of all core API endpoints, the feed to ensure that any user will not

have interruptions in their primary experience on the platform, scrolling through their feed, and the ability for users to upload their images, text, etc. so that they never experience interruptions in creating posts, accounts, etc. Almost any portion of the platform that directly interacts with or affects user experience should remain intact throughout development.

Tools Used: The primary tools used to ensure functionality throughout development are React Testing Library, to ensure frontend functionality; Bruno, to ensure API and backend functionality; and CloudWatch Logs, to ensure backend behavior. More tools used are MySQL Workbench to ensure the database sees the changes it should, AWS Console for more manual testing, and some automated CI/CD testing of the frontend to ensure basic functionality.

5.6 Acceptance Testing

Functional Design Requirements:

Each component must be tested and validated to ensure that the system's functional requirements are met. The API features should be tested to confirm the correct functionality of CRUD operations for users, posts, images, and other related entities. This includes verifying that the Lambda functions triggered by API Gateway interact seamlessly with the database and S3 buckets, as defined in the AWS CDK construct. Using S3 bucket and Cloudfront, the static website hosting solution must be evaluated for accessibility, HTTP redirections, and DNS setup through Route 53. The image optimization features must also be tested to validate image uploads, transformations, and CloudFront caching.

Non-Functional Design Requirements

Non-functional requirements are validated by assessing scalability, security, and maintainability. Scalability is tested by stress testing the APIs and ensuring that CloudFront's caching mechanism optimizes performance under increased loads. Security is reviewed by analyzing IAM policies to verify that resources are only accessible to authorized entities. Maintainability is highlighted through the system's modular architecture, where APIs, static sites, and image optimization are handled separately for better management and updates.

Client Criteria/Involvement

Client involvement begins with presenting test cases that clearly outline how each requirement is validated. Deploying the system in a staging environment allows the client to access and review the static site, API endpoints, and logs. During user acceptance testing, the client is encouraged to interact with the APIs and navigate the static site to confirm the functionality and user experience. Feedback from these interactions will be used to refine our system. The will is also provided with performance metrics, such as the logs from AWS Cloudwatch, and security audit results to show compliance with industry standards. Regular updates through dashboards or progress meetings will show transparency and keep the client informed.

5.7 Security Testing

Requirement: The system must ensure the confidentiality, integrity, and availability of all user data, adhering to best practices and compliance standards for secure application development.

Testing Plan

To validate security compliance, the following tests will be conducted:

- **IAM Policy Review:** Confirm least privilege access for all resources.
- **Penetration Testing:** Simulate attacks to identify vulnerabilities in APIs, S3 buckets, and the network configuration.
- **Encryption Validation:** Verify that all data at rest and in transit is encrypted.
- **Vulnerability Scanning:** Use tools like Amazon Inspector to identify security risks.
- **Access Logging:** Ensure logs are capturing all access events and analyze for anomalies.

Expected Outcomes

- Unauthorized access attempts should be blocked entirely.
- Data breaches or exposures should be mitigated by encryption and strict access policies.
- System components must remain operational and secure under simulated attack scenarios.
- Compliance with industry standards like ISO 27001 and AWS Well-Architected Framework security guidelines should be demonstrable.

5.8 Results

Since testing has not yet begun, there are no direct results to report. However, the foundation for testing is being meticulously prepared to ensure a comprehensive evaluation once the system is ready. The planned testing framework will focus on both functional and non-functional requirements to demonstrate compliance with project goals and user needs.

Planned Testing Approach

The testing strategy is structured to provide both quantitative and qualitative insights into the system's performance. Functional tests will evaluate the core capabilities, such as CRUD operations for users and posts, API Gateway-Lambda interactions, and static website hosting via S3 and CloudFront. Non-functional tests will measure scalability, security, and maintainability, emphasizing metrics like API response times, error rates, and system uptime under stress conditions.

For qualitative insights, user acceptance testing (UAT) will be crucial. This phase will engage stakeholders to validate that the system meets their expectations and effectively addresses their needs.

Demonstrating Compliance

The results from these tests will be analyzed to confirm compliance with the design requirements:

- **Test cases** will verify functional compliance by showing that each feature works as specified.
- **Performance compliance** will involve stress tests to ensure the system handles expected loads and beyond.
- **Security compliance** will include IAM policy reviews and penetration testing to safeguard sensitive data.
- **Usability compliance** will be assessed during UAT, with user feedback informing refinements.

Next Steps

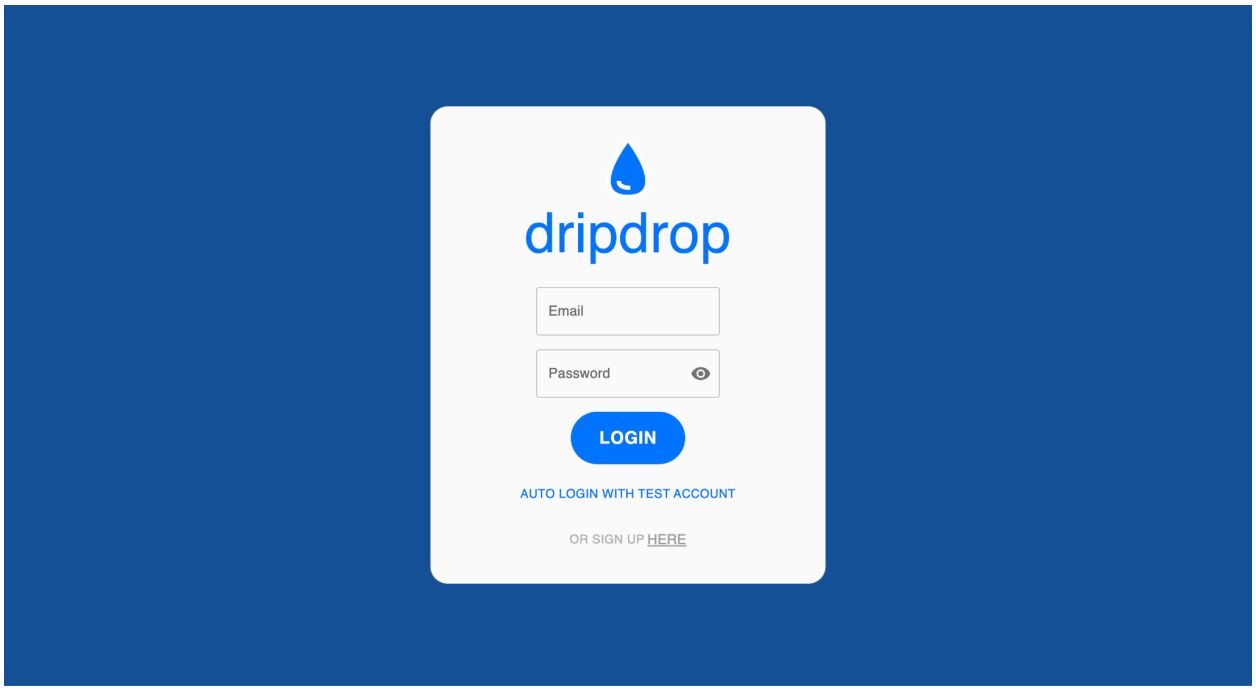
To move forward, the immediate focus is on:

1. Finalizing the test cases and scripts for automated and manual testing.
2. Deploying the system to a staging environment where all components—APIs, static site, and backend services—can be integrated and tested.
3. Gathering client input ensures the testing plan aligns with user expectations and project objectives.

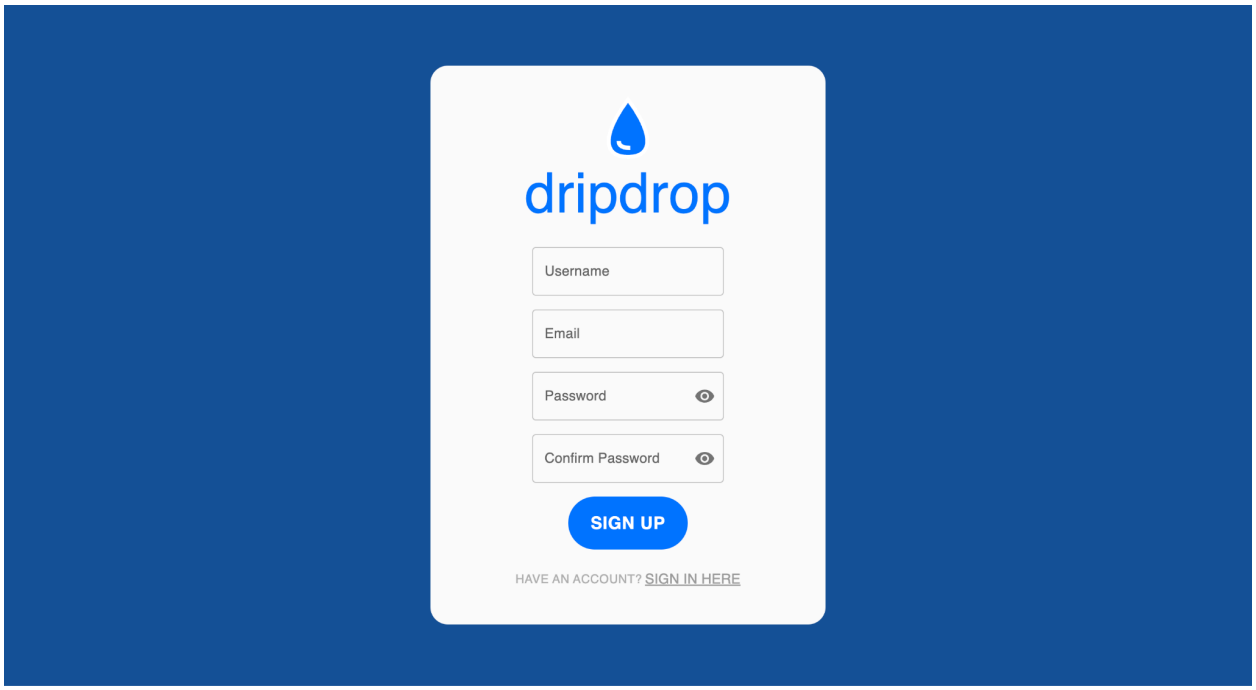
Following this structured approach, the forthcoming testing phase will provide detailed results that substantiate the system's alignment with technical requirements and user needs.

6 Implementation

This semester, we started working on our application's web platform and the backend database to support it. The first aspect of our application that we developed was a preliminary login and sign-up screen. The user can log in with their email address and password. During the signup process, users submit information, including their email address, desired username, and desired password.

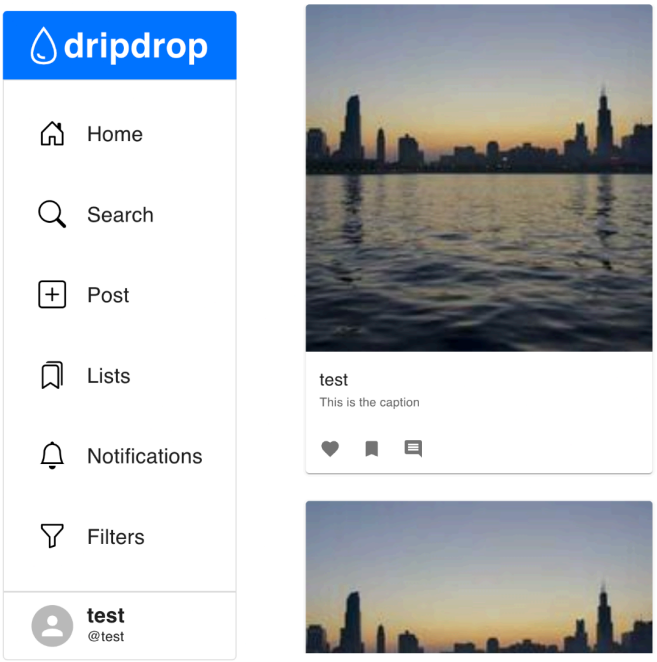


Login Page



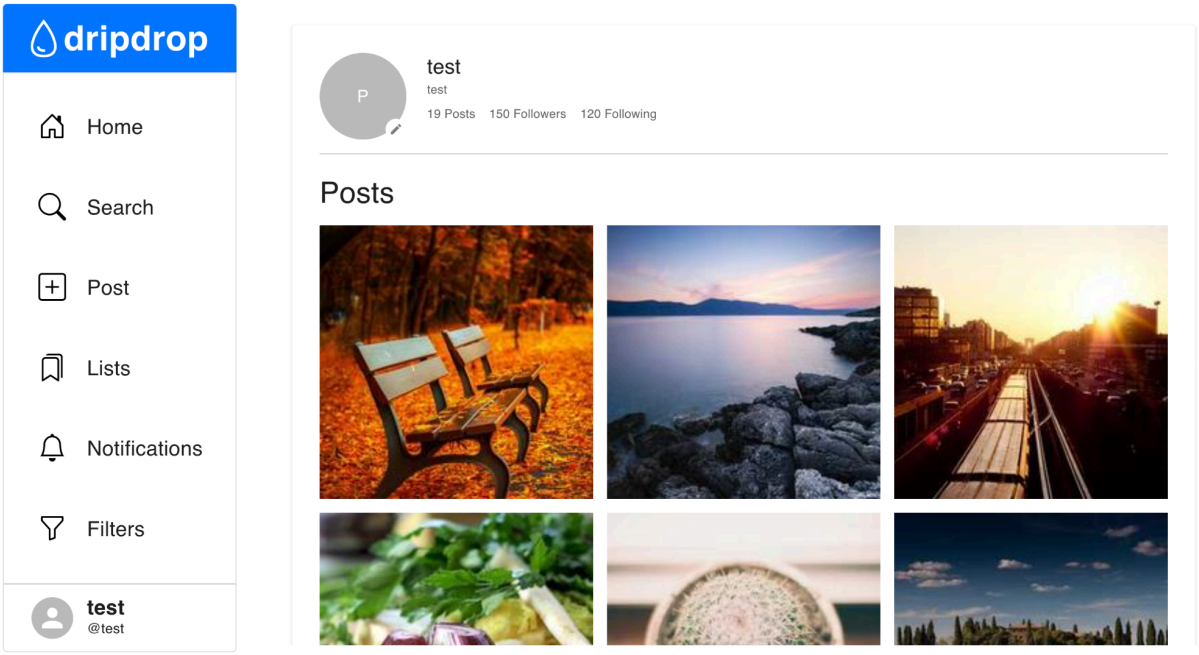
Sign Up Page

Additionally, we have added a home page that displays a feed of posts along with a sidebar that navigates to other pages and features, including user search, notifications, and filtering features, along with pages for posting and viewing lists/collections of saved posts. The posts displayed on the feed currently comprise of a photo, the user who posted it, the caption, and placeholder icons for features to like, bookmark, and comment on the post. These features will be built out early on in the second semester.



Home page

We currently have early implementations of the home page, search bar, and posting page and will work to expand our implementations next semester.
Our final preliminary implementation is the user profile page, which will showcase basic user information, including name, username, profile picture, and follower/following count. Posts from the user can be viewed on this page as well. This page is dynamic and can be accessed through the user search bar or by clicking on the username within a post.



User Profile page

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

This discussion concerns the paper by J. McCormack and colleagues titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

Area of Responsibility	Definition	SE Code of Ethics	How we have addressed this
Work Competence	Work is completed to a high-quality standard and adheres to best practices for software development.	**PRODUCT**: "Ensure that products meet the highest professional standards possible."	We use AWS documentation to conform to their best practices. We use industry standards for our API return statements. More focus will be needed on our quality in the next semester.
Financial Responsibility	Managing our resources efficiently and avoiding unnecessary spending.	**CLIENT AND EMPLOYER**: "Act in the best interests of the client and employer consistent with the public interest."	We communicated with ETG to get our AWS account and budget. Our budget is set in AWS, so we get alerts when we review our monthly target and adjust accordingly.

Communication Honesty	Being transparent and truthful in all communication	**JUDGMENT**: "Maintain integrity and independence in their professional judgment."	We give everyone a chance to speak at our weekly standup, and we are all honest about our contributions.
Health, Safety, and Well-Being	Protecting users from harm and promoting their mental and emotional well-being when using the app.	**PUBLIC**: "Act consistently with the public interest."	At this point we just have a prototype, but we will need to focus on protecting user data and restricting harmful comments/posts next semester.
Property Ownership	Respecting intellectual property rights and acknowledging outside resources we use	**PROFESSION**: "Advance the integrity and reputation of the profession consistent with the public	If we use other people's work, we will make proper attributions to that borrowed work.
Sustainability	Designing the app to be efficient, minimizing resource usage, and supporting long-term maintenance.	**PUBLIC**: "Act consistently with the public interest."	Next semester, one of our big focuses will be on improving the speed and efficiency of our app. At this point we have been focused on just getting an MVP.
Social Responsibility	Considering the app's impact on society, including user privacy and ethical use of data.	**PUBLIC**: "Act consistently with the public interest."	We have some security measures, and we will make thorough security measures once we have completed an MVP for our demo.

Areas we are performing well: Communication, Honesty

Our team exhibits open and transparent communication with each other. We hold regular meetings to discuss progress, share updates, and address challenges. These discussions are a collaborative environment where all team members feel comfortable voicing their concerns and providing feedback. When we meet with our faculty advisor, we are open and honest about our progress updates and don't fake anything.

An area where we can improve: Work Competence

Currently, our team has been focused on learning and growing our skills. There has been a lot of trial and error and emphasis on getting a Minimum Viable Product done by the end of the semester. There has not been a big focus on high-quality work or conforming to standards. As we enter our second semester, we can work on elevating the standard we expect out of our work. We can also improve our prior work to increase the quality. To improve, we should implement more rigorous quality assurance processes, such as automated testing and peer code reviews.

7.2 Four Principles

	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	It's a good social outlet for creative expression that is good for mental health.	Avoiding addictive algorithms that encourage doom-scrolling	Allowing users to choose who their profile and posts are shown to to protect privacy and safety	Making sure everyone's usage of the app is safe for their mental health.
Global, cultural, and social	Allows users to share their sense of fashion with the world and inspire others	Not allowing hateful comments and speech to promote a safe and positive social environment	Users will get a feed tailored to their interests, and they can choose which posts they want to interact with	Ensuring everyone on the app is treated kindly and with respect
Environmental	Encourages users to shop for and discover environmentally healthy and sustainable products	Avoiding business practices that negatively contribute to the environment	Allowing users to find environmentally conscious clothing articles	Giving everyone opportunities to choose from sustainable materials and clothing articles.
Economic	Allows consumers to find cheaper, similar outfits quicker	We avoid subscriptions that suck people in	Giving consumers more budget-friendly options to choose from	Everyone can earn dripdrop points from getting people to purchase apparel items

Where our project shines: Beneficence + Economic

Our app's core feature and purpose is the ability for clothing item comparisons and suggestions. When users see an outfit they like, our app improves their shopping experience by showing them similar items and helping them save money by showing lower-priced items. When a user shares an outfit that they enjoy wearing, they can share it with the world for more people to also enjoy.

Where our project is lacking: Nonmaleficence + Public health, safety, and well-being

For a social media app to perform well, there must be a draw to keep users returning. Many social media apps thrive on addictive doom scrolling. It will be challenging to balance creating

an app that promotes much user interaction but doesn't create addictive tendencies. To improve in this area, we could add a feature that checks in with users when they hit usage benchmarks that are considered unhealthy.

7.3 Virtues

Our Team's Important Virtues:

Cooperativeness - To support cooperativeness, we regularly meet as a team and communicate often in discord. Our meetings are very productive and everyone gets to contribute to the discussion. We leverage each person's strengths when we divide up our tasks.

Responsibility - To support this virtue, we set expectations for when we want to complete assignments, and we break up the tasks among team members. Everyone is dependable and gets their work done on time.

Respect - This virtue is very important to our team because we have a lot of very bright members on our team with varying strengths. Disrespect is not tolerated on our team, and all discussions is constructive and supportive.

Individual Virtues we have demonstrated:

Kolby - Clear Communication and Documentation

This virtue is important to me because it is very crucial when working in team settings. It can have a significant impact on how much a team can accomplish. Documenting and communicating effectively and often can make the lives of everyone on the team much easier. I have demonstrated this virtue by creating documentation for the areas I have worked on, regularly providing updates on Discord, and scheduling/organizing meetings.

Logan - Commitment to Quality

Quality is a crucial aspect of any project, as putting effort into creating quality work from the beginning can prevent future headaches. It can also directly impact the motivation of a team if they see that others are not putting forth quality wor,k which can lead to them not putting forth quality work, resulting in lower-quality work all around. Focusing on quality work upfront can prevent future issues, motivate teammates, and promote a better end result. I have demonstrated this virtue by ensuring that all of the work I do on the project is of high quality and is approved by most of the team.

Kaden - Commitment to quality

Ensuring that a product is completed efficiently and to a high standard of quality has been a core principle I've strived to uphold throughout this course. In a team environment, where multiple people contribute to the same code and documentation, it's essential to produce work

that is clear, collaborative, and of a standard others are proud to endorse. One guiding quote resonates with me: "*How you do anything is how you do everything."* This philosophy reflects my belief that whether it's a small writing task or a large coding project, maintaining the highest level of quality is always crucial.

Zach - Collaboration

The idea for our project was not decided all at once but over time through listening to the ideas of teammates, teachers, and faculty mentors. Through collaborating and considering various ideas and perspectives, I took on a lead role in fine-tuning the initial project idea from the beginning of the semester down to the current idea of *dripdrop*. Collaboration and allowing open thought helped us construct our app idea and have been a crucial point of development throughout the semester, as often multiple team members must work together on parts of the project. I have worked on both the backend and frontend so far, and for both portions of work I have collaborated with other team members to learn from each other and ensure consistency of coding practices.

Gavin - Commitment to intuitive user experience

The most important aspect of any application is the user experience and its clearness. If a design is unclear and unintuitive, users are often frustrated and confused and less likely to continue using the app. That is why I have taken on a significant role in designing the user interface in a way that's both visually attractive and easy to understand. Along with having an intuitive interface, it was also important to me that our app is performant and quick, which goes along with the virtue of commitment to quality.

Elyse - Commitment to objectivity

In designing and creating cloud infrastructure on AWS for our group project, my commitment to objectivity remains a core guiding principle. I focus on evidence-based decision-making, using data-driven insights to ensure our architectural choices meet the technical goals and requirements of the project. By adhering to standardized best practices, such as the AWS Well-Architected Framework, we prioritize security, reliability, performance efficiency, and cost optimization. Setting aside personal preferences, I work collaboratively with the group to evaluate options impartially and align decisions with the overall objectives of the project. This objective approach improves teamwork, ensures fairness, and helps us create a robust and scalable infrastructure.

Individual Virtues we have not demonstrated:

Kolby - Commitment to Objectivity

This virtue is important to me because I know everyone on this team has a lot of great ideas and knowledge and it is important to treat all ideas without my preconceived opinions. To demonstrate this, I will focus on hearing everyone out thoroughly, and not tune out when ideas that conflict with my opinions are voiced.

Logan - Clear and Thorough Documentation

Clear and thorough documentation is paramount to everyone's understanding. Given proper documentation, one can easily go through and understand what someone else created with minimal time and effort on their part. However, without proper documentation, it can be hard for someone who is not well versed in the field to understand resulting in more time and effort wasted. I believe that, moving forward, I need to focus more on providing proper documentation on the work that I complete to ensure everyone has a proper understanding.

Kaden - Techno-social Sensitivity

Our group formed before the semester even began, and we submitted a custom project that aimed to create a Chrome extension for finding discounts through data scraping. However, realized two to three weeks into the semester that similar solutions already existed, and we needed a way to differentiate our project, leading us to pivot. Greater awareness of existing products could have allowed us to identify this issue earlier, saving time and resources. This experience showed me the importance of conducting thorough research in the competitive landscape to understand user needs better, ensuring projects are both innovative and impactful.

Zach - Continuous Improvement

One virtue I need to work on moving forward is continuous improvement. The best applications in the market are not always great right when released, but the initial release is a foundation that can be built on. Instead of programming in a way that simply tries to get the functionality right the first time, I need to integrate into my code the ability to adapt over time. This includes ensuring best coding practices are followed so that code can easily be read and built upon over time. It also means creating functions in a way that is easily repeatable and universal so that a consistent structure can be followed as the app grows and improves.

Gavin - Clear and thorough documentation

One of the virtues that I haven't sufficiently utilized this semester and need to improve on next semester is the virtue of having clear and thorough documentation for my code and contributions. Oftentimes I overlook that my code might be iterated on by other developers who may be confused with my coding style or reasonings, which slows down the development process. Thus, I need to work on adding comments to my code and explaining the limitations and proper uses/implementations of my code and components that I may develop next semester.

Elyse - Clear and Thorough Documentation

I often find myself struggling with the virtue of clear and thorough documentation, as I tend to prioritize completing tasks over documenting the processes and decisions behind them. While I understand the importance of detailed documentation in improving reproducibility, clarity, and ease of collaboration, I sometimes find it challenging to balance this with the technical demands of the project. This can lead to gaps in understanding when refactoring previous work or explaining decisions to team members. To improve, I plan to integrate documentation as an active step in my workflow rather than treating it as an afterthought.

8 Closing Material

8.1 Conclusion

Summarizing what we have done so far, our team currently has a working AWS Aurora server hosting our database and website. We have a primarily functional backend, including all of the needed lambdas for user functionality. Our team has a frontend that allows users to view their feeds, create posts, and search user profiles. Additionally, the team currently has a working AI model that can tag different attributes of apparel items in an image and recognize the items apart from each other.

Our goal is to have a working website with the basic functionalities by the end of this semester. Additionally, we want a working mobile application by March of next semester and a working AI outfit-matching model by the end of next semester. So far, the main constraints have been finding the time between all the other commitments as a college student to develop an entire application. Additionally, there are a lot of new concepts for our team to learn, the the development has been a slower process.

As the team progresses on development, we will continue focusing on continuous improvement and following Agile methodologies. For the design, we will transition next semester from focusing mainly on functionality - as we primarily focused on this semester - to additionally focusing heavily on application speed and usability.

8.2 References

AWS Official Documentation

Amazon Web Services, "AWS Documentation," [Online]. Available: https://docs.aws.amazon.com/. [Accessed: Dec. 07, 2024].

TypeScript Official Documentation

Microsoft Corporation, "TypeScript Documentation," 2024. [Online]. Available: https://www.typescriptlang.org/docs/. [Accessed: Dec. 7, 2024].

Python Official Documentation

Python Software Foundation, "Python Documentation," 2024. [Online]. Available: https://docs.python.org/. [Accessed: Dec. 7, 2024].

GitLab Blog Article

GitLab Inc., "How to Deploy a React Application to Amazon S3," GitLab Blog, Mar. 1, 2023. [Online]. Available: https://about.gitlab.com/blog/2023/03/01/how-to-deploy-react-to-amazon-s3/. [Accessed: Dec. 7, 2024].

Material-UI Official Documentation

MUI, "Material-UI: React Components for Faster and Easier Web Development," 2024. [Online]. Available: https://mui.com/material-ui/. [Accessed: Dec. 7, 2024].

PyTorch Vision Documentation

PyTorch, "TorchVision: PyTorch's Computer Vision Library," 2024. [Online]. Available: https://pytorch.org/vision/stable/index.html. [Accessed: Dec. 7, 2024].

Ultralytics Documentation

Ultralytics, "Ultralytics Documentation," 2024. [Online]. Available: https://docs.ultralytics.com/. [Accessed: Dec. 7, 2024].

IDEALS Professional Responsibility

J. McCormack, "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment," *Int. J. Eng. Educ.*, vol. 28, no. 2, pp. 416–424, 2012.

9 Team

9.1 Team Members

1) Kaden Wingert
2) Kolby Kucera
3) Zachary Foote
4) Logan Roe
5) Elyse Kriegel
6) Gavin Rich

9.2 Required Skill Sets for the Project

Software Development Skills:

Frontend: Experience with frontend programming, specifically with React, typescript, and CSS, will be essential for creating the pages for the frontend. These skills are needed to create a polished frontend that can display a smooth interface for users.

Backend: The project requires the ability to create backend Python files that will interface with the database. Knowledge of creating backend lambdas that can be called at an API endpoint is needed.

API: The team's developers will need to be able to create APIs that connect the frontend with the back end. Full-stack web development knowledge will be required within the team to understand how the APIs work and how to implement them properly.

SQL: SQL skills will be needed to interface with our MySQL database. Knowledge of how to query and what information can be stored in an SQL database will be needed.

Amazon Web Service (AWS): AWS knowledge will be needed to host both the mySQL database (for the storage of regular items e.g., account info, posts, passwords, and usernames) and S2 bucket items (images). To set this up in a proper and cost effective way and to know which resources would be the best for our team to utilize, the team will need experience with AWS.

AI-Model Creation Skills: AI model skills and experience will be needed to create the view similar items page. The team is working to create an AI model from scratch that can tag the photos based on the traits of the clothing items in the image. To do this, the team will need experience with working with AI models so that there is a foundation of knowledge to build off of in the development of Dripdrop's model.

Entrepreneurship Skills: The team also requires entrepreneurship skills. This is because the team will need to pitch the idea to large retail companies to create partnerships, an essential part of the dripdrops business model. Additionally, the team will need to create an initial user

base for the app, and the best way to do this is by speaking to people and convincing them that dripdrop will benefit them to use.

Leadership Skills: Leadership skills are required for the team to have effective meetings and communication. Individuals who can effectively keep the group's tasks and goals in check and take the lead in group meetings allow the entire group to function effectively and thoroughly.

9.3 Skill Sets Covered by the Team

Software Development Skills:

Frontend: Logan, Zach, Gavin, Kaden

Backend: Elyse, Kolby, Kaden, Gavin

API: Elyse, Kaden

SQL: Elyse, Kaden, Logan

AWS: Elyse, Kaden, Kolby

AI-Model Creation Skills: Elyse

Entrepreneurship Skills: Zach

Leadership Skills: Kolby

9.4 Project Management Style Adopted by the Team

Our team uses an Agile approach to project management. All tasks are placed onto a Git Task Board. We have weekly meetings highlighting what we accomplished and what we plan to do in the upcoming weeks. We chose a meeting lead, Kolby Kucera, who leads the team and keeps us on track in these weekly meetings. During the meetings, the team also moves tasks on the task board to determine what it completed, in progress, or saved for future week's work. As a team, the decisive goal of these meetings is to ensure that we are constantly iterating and making minor changes to our project.

We also assigned other team leadership roles based on the individual's strengths and interests. These other roles include a communication lead, a research lead, a project manager, a technical lead, and a testing lead.

9.5 Initial Project Management Roles

Team Liaison/Communication Leads: Zachary Foote

Research Lead: Logan Roe
Project Manager: Kaden Wingert
Technical Lead/Architect: Elyse Kriegel
Testing/Quality Assurance: Gavin Rich
Documentation/Meeting Lead: Kolby Kucera

9.6 Team Contract

Team Members:

1) Kaden Wingert
2) Kolby Kucera
3) Zachary Foote
4) Logan Roe
5) Elyse Kriegel
6) Gavin Rich

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
   Weekly Meetings every Thursday from 1:50 - 2:50 pm in person at the Student Innovation Center.
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
   Discord will be our primary method of communication. Snapchat/Phone is used for emergencies. We will store documents in our shared Google Drive folder.
3. Decision-making policy (e.g., consensus, majority vote):
   When making decisions, we will vote to determine our course of action. A minimum of 4 out of 6 people will have to agree.
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
   Kolby will take notes during meetings and upload the notes to our shared Google Drive after each meeting.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
   Unless otherwise communicated, all group members must attend all team meetings on time. If someone is going to be absent or late, they must communicate in advance and let the team know how they will make up their missed time.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

All group members should contribute their fair share in all assignments, timelines, and deadlines. We will fairly and evenly assign tasks to each team member for each assignment.

3. Expected level of communication with other team members:
   Group members should respond to messages within a day (barring unforeseen circumstances). The group members are expected to complete their assignments on time and frequently communicate any comments, questions, or information that the rest of the group should know.
4. Expected level of commitment to team decisions and tasks:
   We expect all members to be committed and enthusiastic about our project. This involves actively participating in the weekly group meetings and completing any of their assigned tasks. If a member is not contributing, we will address it at our weekly meeting.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
   Team Liaison/Communication Leads: Zachary Foote
   Research Lead: Logan Roe
   Project Manager: Kaden Wingert
   Technical Lead/Architect: Elyse Kriegel
   Testing/Quality Assurance: Gavin Rich
   Documentation/Meeting Lead: Kolby Kucera
   Full-stack Developer: All members will be developing both frontend and backend

2. Strategies for supporting and guiding the work of all team members:
   We will frequently do progress check-ins with the group members through Discord to see how everyone is doing on their tasks. This will facilitate conversations if they are struggling and need help from the group. We will also hold brief stand-up meetings at the start of our weekly meetings. These stand-up meetings will address any challenges being faced, and the group will work to address this.
3. Strategies for recognizing the contributions of all team members:
   We will regularly highlight individual achievements and contributions during team meetings. This would involve shouting out successful project completions, creative

problem-solving, or efforts that align with the team's goals. We will implement regular project checkpoint retrospectives where team members can reflect on each other's contributions. This not only encourages recognition but also improves collaboration and team dynamics.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
   Kaden: Experience developing in C, C++, Python, Java, web development (React, JavaScript, HTML, CSS, Typescript), mobile app development, AWS, SQL, MongoDB.
   Kolby: Experience with full stack development on ERP systems and web applications. Skills: C, Python, Java, Relational Databases, SQL, HTML, CSS, Javascript, Typescript, AWS
   Zachary: Experience with Front end programming (HTML, CSS, Javascript, Java, Android Studio). Experience with retrofit to allow frontend-backend communication.
   Elyse: Typescript, Python, Java, HTML, CSS, React, AWS, SQL, Kotlin, C, C++, C#
   Logan: Experience developing in Java, JavaScript, HTML, CSS, MATLAB, C++, Python, React, Typescript, android app development, and SQL.
   Gavin: Experience with full-stack development within the ASP.NET environment and C#. Other development experience in React, Python, Java, HTML/CSS/JavaScript, SQL, and C.
2. Strategies for encouraging and supporting contributions and ideas from all team members:
   We will use inclusive language and encourage open discussions to foster an environment where every voice is valued. Additionally, we will encourage brainstorming sessions where any idea is considered valid for discussion. Our stand-up meetings and retrospectives will be a way to give each team member an equal opportunity to share updates and concerns in a structured way.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment obstructs their opportunity or ability to contribute?)
   We will address an issue if a team member brings it up during our weekly meeting. If necessary, we will schedule an emergency meeting to discuss their concerns or discuss

them at our weekly meetings. Periodically, we will assess how the group feels about collaboration and inclusion. This can be done through structured reflections during team meetings.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    - A working prototype of a functional Chrome extension is ready for demo.
    - Complete design of all of the pages on the website and app on an application such as Figma.
    - A clear vision of roles for the developmental phase of the project.
    - Incremental progress from each teammate (weekly updates on work).
    - Backend capable of pulling prices from 3 or more websites.

2. Strategies for planning and assigning individual and teamwork:
   We will create stories on our GitLab Kanban board every week at our stand-up meeting and whenever they are needed. These stories will be assigned to members to evenly distribute work as well as a way to keep track of what still needs to be done. We also will keep a product backlog where we can have extra features that are lower priority. This allows the developers to pick up tasks if they finish their tasks earlier than anticipated.

3. Strategies for keeping on task:
   We will break the project into milestones with clear deadlines. Tools such as Gantt charts can be helpful to visualize the timeline of the project. Our weekly standup meetings (15 minutes max) will allow each team member to share their progress, what they'll be working on next, and any issues they are having.
   After our standup meeting, we will develop a plan of what we want to accomplish with the remaining meeting time. This will give us a short-term goal to keep us focused on the task.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

When a team member violates a part of the team contract by disrespecting a team member, failing to complete their tasks on time, not showing up to meetings, etc, we will immediately identify the issue. We will allow them to explain their situation and any challenges they may face. Then, we will remind them of this team contract and their agreed-upon responsibilities. Next, we will set goals for improvement to ensure the team members know the consequences of repeated infractions.

2. What will your team do if the infractions continue?

   If infractions occur, we will record the specific incidents, including dates, the nature of the issues, and any attempts to resolve them. If the situation doesn't improve after following the previously outlined steps, we will schedule a meeting with the professor as an intermediary to discuss issues. We will present the documentation of the infractions and describe the steps we took to address them.

   The professor can offer a different perspective and guide in handling the situation.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*
b) *I understand that I am obligated to abide by these terms and conditions.*
c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Kaden Wingert DATE 9/17/2024
2) Kolby Kucera DATE 9/17/2024
3) Logan Roe DATE 9/17/2024
4) Gavin Rich DATE 9/19/2024
5) Zachary Foote DATE 9/19/2024
6) Elyse Kriegel DATE 9/19/2024