



dripdrop

Design Document

Team 25

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

Faculty Advisor: Simantra Mitra

Email: 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 outfit posts 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 achieve a similar look, but for a better price.

The key design requirements include a website, a mobile application, and a working AI-outfit matching model. A smooth and intuitive user interface for both the website and the application is crucial to provide 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 assisted by AI, search accounts, and receive AI product recommendations.

The architectural design of this application includes a MySQL database and an S3 bucket hosted on Amazon Web Services. The MySQL database is used for data retrieval and storage, and the s3 bucket is used to store the images. A Python CDK (Cloud Development Kit) was used to provision the entire infrastructure, including the creation of lambda functions for the backend to ensure the app is efficient and cost-effective. The front end of the website leverages React, while the mobile app is written with React-Native and the Expo library.

Our design effectively meets the initial requirements by providing a fully functional mobile app and website where users can create an account, follow other users, like and comment on posts, create posts assisted by AI, search accounts, and receive AI product recommendations. Additionally, our AI-driven outfit matching model successfully suggests similar styles to users, enhancing affordability and accessibility in fashion. While the system is fully implemented, we have not yet performed formal user acceptance testing (UAT) to validate the overall user experience and ensure the design meets all usability expectations. Future testing phases will be necessary to gather direct user feedback and refine any UI/UX inconsistencies.

Future iterations would include deeper AI personalization, enhanced filtering for recommendations, and partnerships with brands to create the affiliate ecosystem. Scaling considerations include optimizing AWS resources to support higher user loads. Furthermore, with additional time, a more thorough set of tests would be implemented including unit tests, integration tests with at least 80-90% code coverage.

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 and Android app
 - User authentication
 - Posts
 - User profile and feed
 - AI product suggestions
- Backend
 - MySQL Database
 - API Endpoints
 - Backend CRUD operation and business logic functions
 - Storing images
- Infrastructure
 - AWS CDK code to setup/manage the infrastructure
 - Host database
 - Host frontend code
 - CI/CD pipeline
- AI
 - Obtain comprehensive dataset
 - Creating an AI 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 Services: CDK (Cloud Development Kit), IAM (Identity and Access Management), Lambda, RDS (Relational Database Service), API Gateway, Aurora, Route 53, Secrets Manager, EC2 (Elastic Cloud Compute) Instance, Sagemaker
- Python
- Typescript
- React Native
- Bruno - API client
- React-Native
- Expo - React Native framework
- Creating and training an AI model
 - Pytorch for model creation
 - Nvidia cuda for training/inference
- OpenCv for image processing
- Pillow for image transformation

Table of Contents

Executive Summary.....	2
Learning Summary.....	3
Development Standards & Practices.....	3
Summary of Requirements.....	3
Applicable Courses from Iowa State University Curriculum.....	4
New Skills/Knowledge acquired that was not taught in courses:.....	4
Definitions and Key Concepts.....	8
1. Introduction.....	10
1.1 Problem Statement.....	10
1.2 Intended Users.....	10
2. Requirements, Constraints, And Standards.....	12
2.1 Requirements and Constraints.....	12
2.1.1 Functional Requirements.....	12
2.1.2 Non-Functional Requirements.....	12
2.1.3 Infrastructure Requirements.....	13
2.1.4 Aesthetic Requirements.....	13
2.1.5 User Experiential Requirements.....	13
2.2 Engineering Standards.....	14
2.2.1 Importance of Engineering Standards.....	14
2.2.2 IEEE Standards applicable to dripdrop.....	14
2.2.3 Rationale for Applicability.....	15
2.2.4 Other Standards.....	16
2.2.5 Modifications needed to incorporate these standards.....	16
3 Project Plan.....	17
3.1 Project Management and Tracking Procedures.....	17
3.2 Task Decomposition.....	18
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria.....	19
Milestone 1 - Website.....	19
Milestone 2- Mobile Application.....	20
Milestone 3 - AI Similar Product Suggestions.....	20
3.4 Project Timeline/Schedule.....	20
3.5 Risks and Risk Management/Mitigation.....	22
3.5.1 Frontend.....	22
3.5.2 Backend.....	22
3.5.3 Infrastructure.....	23
3.5.4 Artificial Intelligence.....	23
3.5.5 Risks That Came to Pass.....	24
3.6 Personnel Effort Requirements.....	26
3.6.1 Semester 1 Estimate.....	26

3.6.2 Actual Personnel Effort Requirements.....	26
3.7 Other Resource Requirements.....	27
4. Design.....	28
4.1 Design Context.....	28
4.1.1 Broader Context.....	28
4.1.2 Prior Work/Solutions.....	28
4.1.3 Technical Complexity.....	29
4.2 Design Exploration.....	30
4.2.1 Design Decisions.....	30
4.2.2 Ideation.....	31
Infrastructure/Server Provider.....	31
4.2.3 Decision-Making and Trade-Off.....	31
4.3 Final Design.....	32
4.3.2 Detailed Design and Visuals.....	33
4.3.2.1 API High-Level Overview.....	33
4.3.2.2 Static Website Hosting.....	35
4.3.2.3 Image Optimization Stack (CDN).....	38
4.3.2.4 Image AI Processing Stack.....	41
4.3.3 Functionality.....	43
4.3.4 Areas of Challenge.....	44
4.4 Technology Considerations.....	45
Cloud Provider: Amazon Web Services (AWS).....	45
Database: MySQL on Amazon Aurora.....	46
Frontend Technology: React.....	46
5 Testing.....	47
5.1 Integration Testing.....	47
5.2 Frontend Testing.....	48
5.3 User Testing.....	48
5.4 Unit Testing.....	48
5.5 Interface Testing.....	49
5.6 System Testing.....	49
5.7 Regression Testing.....	49
5.8 Acceptance Testing.....	50
5.9 Results.....	50
6 Implementation.....	51
6.1 Frontend.....	51
6.1.1 Mobile Application.....	51
6.1.2 Website.....	61
6.2 Backend.....	63
6.3 AI Model.....	64
6.4 Security.....	65

6.5 Reflection - What worked and what didn't.....	66
7 Ethics and Professional Responsibility.....	68
7.1 Areas of Professional Responsibility/Codes of Ethics.....	68
7.2 Four Principles.....	69
7.3 Virtues.....	70
8 Closing Material.....	73
8.1 Summary Of Progress.....	73
8.2 Value Provided.....	74
8.3 Next Steps.....	74
1. Beta Testing with a Small Group of Users.....	74
2. Increased Testing Efforts.....	75
3. Improving AI Autofill and Clothing Item Segmentation.....	75
4. Developing a Marketplace Feature.....	75
9 References.....	76
Appendix 1 - Operation Manual.....	78
Appendix 2 - Alternative/Initial Version of Design.....	78
1. Initial Version: Chrome Extension for Finding the Best Deals.....	78
2. Shift to Mobile App Development.....	79
3. Removal of Affiliate Link Feature.....	79
Appendix 3 - Code.....	79
Appendix 4 - Team Contract.....	79
Team Members.....	79
Required Skill Sets for the Project.....	79
Skill Sets Covered by the Team.....	80
Project Management Style Adopted by the Team.....	81
Initial Project Management Roles.....	81
Team Contract.....	81

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. **Segmentation:** the process of which the AI distinguishes between different clothing items
27. **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.
28. **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.
29. **Unit Testing:** Testing individual software components to ensure they perform as expected.
30. **User Personas:** Fictional representations of target users based on demographics, needs, and behaviors to guide design and development.
31. **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 a unique, valid email address and a unique username.
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 posts and like them, comment on them, and save them to come back to later.
4. Users must be able to see “markers” on all posts that contain information about the clothing item and can be used to see similar clothing items.
5. Users need the ability to upload images of their clothing with descriptions (brand, price, size, etc.)
6. Users must be able to add/remove/edit markers on their own posts, including editing clothing descriptions.
7. The app must automatically suggest tags based on image recognition and AI (e.g., "denim jacket," "red shoes").
8. The app must allow users to search for specific user accounts and/or posts.
9. The system must use AWS S3 to handle user-uploaded media.
10. The system must use AWS S3 for static website hosting.
11. Users must be able to reset their password using their email address.
12. The system must leverage AI to recommend similar clothing items based on an image of a clothing item.

2.1.2 Non-Functional Requirements

1. The system must use Amazon Aurora as a scalable database.
2. The database must support replication across multiple availability zones (AZs) to ensure high availability and fault tolerance.
3. Implement role-based access control (RBAC) to define user roles and permissions, ensuring restricted access to sensitive actions or data.
4. The system must use AWS CloudFront to cache and serve content (static files and images) globally, reducing latency for users in different regions.
5. All user requests, aside from AI-based requests, should be completed within 500 ms of when it was requested.
6. The app should not allow for any user to sign up if under thirteen years old.
7. API calls may only be made by verified users, which is ensured via AWS Cognito Authentication.
8. Due to the use of AWS Lambdas for performing API calls, the application should be able to handle 1,000 concurrent sessions.
9. The application should be simple and intuitive to navigate.

2.1.3 Infrastructure Requirements

1. GitLab Actions must be used for the CI/CD (Continuous Integration & Continuous Development) pipeline to automate the building, testing, and deployment of code.
2. AWS CDK must be used to oversee and manage cloud infrastructure.
3. AWS Cost Explorer must be used to monitor AWS costs and ensure that the platform stays within the monthly budget.

2.1.4 Aesthetic Requirements

1. The application must be navigable and intuitive enough for any new users to create a post in under 90 seconds.
2. The system must use legible, modern fonts with proper size and spacing to ensure content is easy to read on all screen sizes.
3. Understandable and standardized icons must be used to convey information to the user in a straightforward format.

2.1.5 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

Agile: 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. It 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

High Level: 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.

Backend: 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.

Frontend: 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 saved posts so users can revisit specific posts or clothing items later.

Infrastructure: 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.

Artificial Intelligence: 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.

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.

Frontend: 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.

API: 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.

Backend: 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. 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 Similar Product Suggestions

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

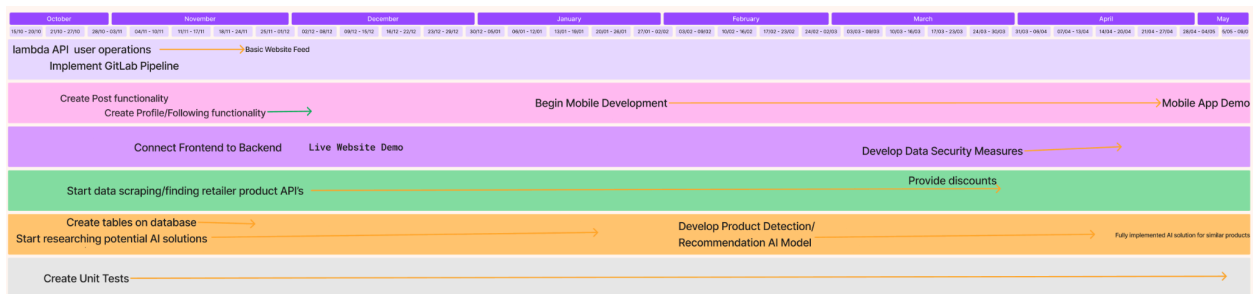


Figure 1: First Semester Gantt Chart

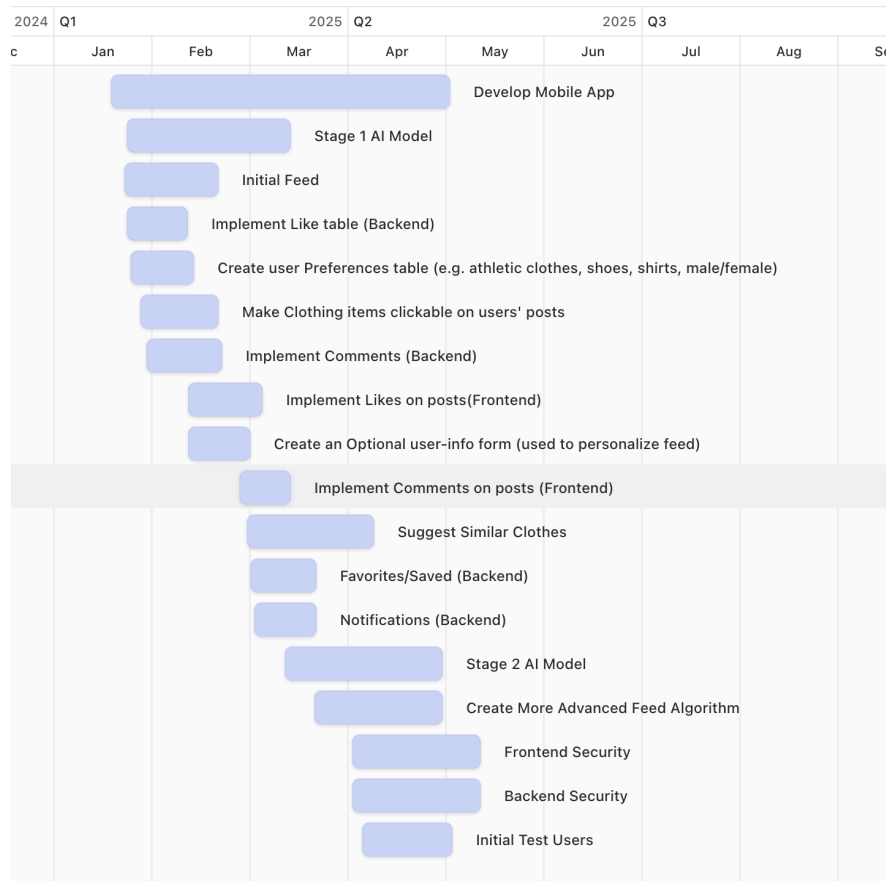


Figure 2: Second Semester Gantt Chart

As described above, our project has three major milestones, each with several subtasks and timelines:

Website: 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.

Mobile Application: 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.

AI Model: 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.

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

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.

3.5.5 Risks That Came to Pass

1. User Authentication Security Vulnerabilities

- Occurred: We, originally, did not limit API access using AWS Cognito, or similar software, so that left us vulnerable to people performing unauthorized API calls which could lead to loss of data and privacy concerns.
- Response: We implemented AWS Cognito Authentication so that only verified users could make API calls. Therefore, API calls could no longer be called by just anyone, but only verified users on the application, through the application, could make API calls.

2. User Profile Being Unintuitive

- Occurred: The user profile was unintuitive and was lacking important functionality early on due the complexity and sheer amount of information we needed to display.
- Response: We decided on ways to mitigate the number of different “sections” we needed to decrease the complexity, such as merging “Needs Review” posts and “Draft” posts into just “Drafts.” Further, we used more appropriate and common words to describe certain sections so it was more recognizable and easier to understand.

3. High Complexity in Table Relationships

- Occurred: Early database models required a significant amount of joins and nested queries which increased the complexity for the frontend as they had to make multiple calls to perform actions that should be done in one API call.
- Response: We revised our schema to simplify joins and also used indexing to improve load speeds for more complex queries. Further, we revised some of our table relationships to have them be more simplistic, where possible, to improve our original, naive approach.

4. Lambda Functions Breaking After Updates

- Occurred: Due to changing ideas and complex updates, we ran into lambda functions breaking the frontend on updates. Early on, this led to numerous unexpected issues for the frontend developers.
- Response: We began to more clearly communicate the changes prior to implementing them so that frontend developers would have a better understanding and idea of what

was going to happen when. This allowed us to respond quicker and more reasonably to unexpected issues with the updates and for unexpected issues to become expected on the frontend.

5. Changing Plans Leading to Database Management Issues

- Occurred: Due to the nature of software development, where rapid iteration and design changes occur often, we had significant changes arise to our plans which lead to database changes. This caused many database drops, sometimes unexpectedly, which could lead to interruptions in the workflow.
- Response: We more clearly communicated when would be a good time to drop the database, why it was being dropped, and when we actually decided to drop it, so everyone knew what to expect and when to expect it.

6. iOS vs. Android Differences

- Occurred: Due to the nature of the different OS's, we ran into numerous issues where the frontend would perform as expected on Android but not on iOS, or vice versa. This led to many frustrations as people would run into issues that the original developer did not experience, due to an OS difference.
- Response: When we had larger changes/updates, especially ones that are known to have different behavior between iOS and Android, we would have someone from the opposing OS test it prior to merging to master. This would lead to significantly reduced unexpected breaks due to OS differences.

7. Unmonitored AWS Spending

- Occurred: Due to wanting a fast and responsive AI workflow, we originally used AWS SageMaker to perform AI tasks. This, as well as not regularly monitoring costs, lead to a very unexpectedly high cost month.
- Response: We began checking our costs more regularly, especially when testing something new, and set alerts for different costs to ensure we did not have another incident such as that.

3.6 Personnel Effort Requirements

3.6.1 Semester 1 Estimate

This table provides a structured overview of tasks along with estimated hours for each task in the Frontend, Backend, Infrastructure, and AI sections. The tasks are broken down into very high level categories of tasks, to provide simplicity in the table. The categories roughly follow our list of requirements as discussed above.

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

Figure 3: Estimated work hours chart

3.6.2 Actual Personnel Effort Requirements

The next table expands on our initial table from the first semester, and gives rough estimates for the actual work hours spent on all of our high level tasks. This includes work from both the first and second semester, and keeps the tasks organized at a very high level. In reality, we worked at a very fast and incremental pace, but showing every single task would be a lot harder to visualize than the chart below.

Task	Hours
Frontend	
User Authentication	10
Create Post	40
User Profile	20
Bookmarks	2
Scrollable Feed	40
Search	15
AI Recommendations	10
Backend	
Creating Tables	10
General CRUD Lambdas	35
37 API Endpoints	5
AI Workflow Lambdas	20
AI Recommendations	10
Image Storage	25
Infrastructure	
Hosting Website	5
AWS CDK code	30
Database Management	10
AWS Cognito Security	15
Pipeline	20
AI Model	
Training	50
AI Workflow	40
Total	
	412

Figure 3.1: Actual Work Hours Chart

3.7 Other Resource Requirements

The primary resource supporting our project is our AWS account. We worked with ETG to secure a shared AWS account for the team and submitted a monthly budget projection, which was approved. AWS currently supports our cloud-based infrastructure, including data storage, backend services, and real-time model inference.

In addition to AWS, we leveraged Iowa State University's Nova research computing cluster to train our deep learning models. Since none of our team members had access to a high-end GPU suitable for large-scale training, we utilized NVIDIA A100 GPUs on Nova to efficiently train our computer vision models.

Specifically, we trained YOLO-based segmentation and classification models using the Fashionpedia dataset, which contains detailed annotations for clothing items and attributes. This model is used to automatically identify and label clothing items (e.g., type, color, and style) from user-uploaded images, enabling key features like automated tagging and visual recommendations.

4. Design

4.1 Design Context

4.1.1 Broader Context

Dripdrop targets fashion-conscious users seeking affordable outfit inspiration and a platform to monetize their style. It addresses the following areas:

Area	Description	Examples
Public Health/Safety/Welfare	Encourages sustainable shopping and provides income opportunities for users via affiliate rewards	reducing fast fashion waste
Global/Cultural/Social	Aligns with Gen Z/Millennial digital-native practices	social sharing, influencer culture
Environmental	Promotes conscious consumption by highlighting cheaper alternatives, potentially reducing overproduction	AI-driven recommendations prioritize local/affordable brands, reducing long-distance shipping
Economic	Creates micro-earning opportunities for users and cost-saving alternatives for shoppers.	Brands pay commissions only for converted sales (low-risk marketing).

4.1.2 Prior Work/Solutions

Pinterest : Focuses on inspiration but lacks monetization and AI-driven recommendations. Dripdrop adds affiliate rewards and price-conscious alternatives.

LTK: Enables monetization but requires existing influencer status. Dripdrop democratizes earning for all users.

Amazon Outfit Builder: Recommends items but lacks social features. Dripdrop combines social interaction with AI-powered shopping

4.1.3 Technical Complexity

Our project meets the criteria for technical complexity through a multi-component architecture leveraging AWS cloud services and AI-driven automation, requiring integration across distinct engineering domains.

1. Multi-Component System with AWS

All infrastructure was designed and deployed on AWS, integrating the following subsystems:

- a. Frontend:
 - i. Website (React + S3): Dynamic UI and serverless hosting via S3
 - ii. Mobile (React Native): Compatibility with iOS and Android devices
- b. Backend (API Gateway + Lambda + Aurora MySQL)
 - i. RESTful API built with Lambda functions for CRUD operations (users, posts, rewards).
 - ii. RDS Proxy for efficient database connection pooling under load
 - iii. AI/ML (SageMaker): Image classification, segmentation, and recommendation engine.
- c. AI Recommendations:
 - i. AI analyzes uploaded images to automatically detect item type, dominant color, and style attributes—minimizing the need for manual input.
 - ii. Based on visual similarity (type, color, shared tags), the system recommends similar public posts, optionally favoring more affordable alternatives if desired.
 - iii. Cropped item segments are batch-processed through an AI model to identify relevant clothing attributes like texture, fit, or pattern.
 - iv. If no visually similar items are found, the system relaxes criteria (e.g., drops color match) to ensure relevant suggestions are still provided.

2. Challenging Requirements Exceeded Industry Standards

- a. Cross-Platform Support
- b. AI Recommendations
- c. Scalability and Cost Efficiency: Every piece of infrastructure is hosted on AWS in a CDK which allows for effortless dynamic scaling and code reusability
- d. Serverless Cost Optimization:
 - i. Cost Savings: We pay only for execution time rather than a server running 24/7.
 - ii. Auto-Scaling: AWS Lambda automatically scales to handle incoming requests, provisioning execution environments as needed, up to a default concurrency limit of 1,000 concurrent executions across all functions in an AWS region

4.2 Design Exploration

4.2.1 Design Decisions

- AWS Cloud Infrastructure
 - Why: Scalability, integrated AI/ML tools (SageMaker), and cost-effectiveness.
 - Impact: Enabled seamless deployment and future growth
- Relational Database (Aurora MySQL)
 - Why: Structured data (users, posts, items) with ACID compliance.
 - Impact: Ensured data integrity for transactions (e.g., point rewards).
- React Frontend (Website):
 - Why:
 - Performance: Virtual DOM enables faster UI updates (~50ms render times).
 - Modularity: Component reuse across pages (e.g., shared PostCard for feed/profile).
 - Community: 16M+ weekly npm downloads ensure long-term support.
 - Impact: Accelerated development and seamless integration with many third-party API
- React Native (mobile app):
 - Why:
 - Cross-Platform Efficiency: Single codebase for iOS/Android
 - Hot Reload: Instant UI updates during development.
 - Impact:
 - Faster Time-to-Market: Developed both platforms simultaneously saving reducing mobile development time by 50%
- Why Not React Native For Web As Well?
 - Why:
 - React Native's ecosystem and libraries are primarily focused on mobile, leading to a lack of features and tooling specifically designed for web development
 - React Native's architecture and layout engine are not optimized for the web environment, which can lead to performance issues and compatibility problems
 - Library Support: Critical web libraries (e.g., react-query, Framer Motion) are React-only.
 - Impact:
 - Optimal UX: Achieved sub-1s page loads (vs. 2.5s avg. with RN Web).
 - Feature Parity: Leveraged React's mature ecosystem (e.g., drag-and-drop uploads).

4.2.2 Ideation

Infrastructure/Server Provider

Early on, one of the most significant decisions to be made was how critical aspects of the project would be hosted such as the database and front-end code. As discussed above, the primary decision was Microsoft Azure vs. AWS.

Initially, three potential paths to take were identified: using on-premise Iowa State servers, a cloud provider, or combining the two to employ a hybrid approach. Minimal time was spent exploring more options because we have to use what Iowa State will provide us or find suitable cloud options for servers and compute services. 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.

4.2.3 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 seven 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, AWS obtained the highest score. 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.

4.3 Final Design

4.3.1 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.

4.3.2 Detailed Design and Visuals

4.3.2.1 API High-Level Overview

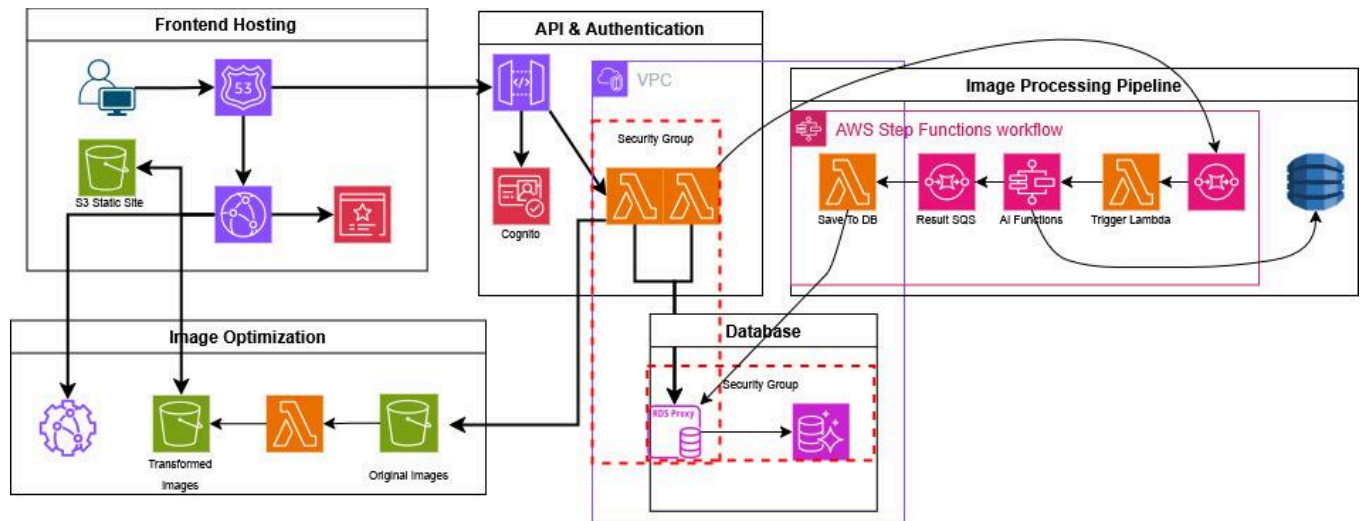


Figure 4: API Architecture Diagram

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.

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

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.

4.3.2.2 Static Website Hosting

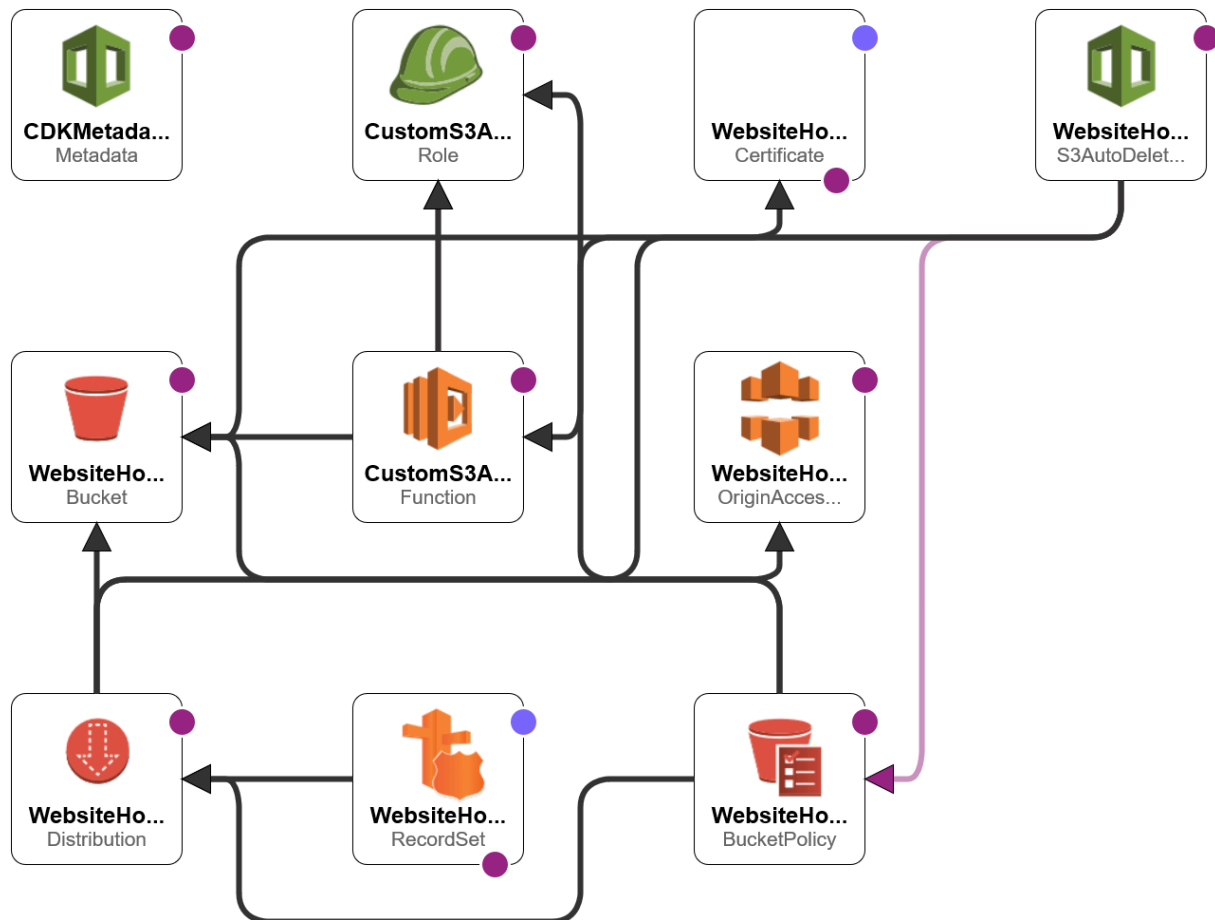


Figure 5: Website Architecture Diagram

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.

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.

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.

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.

4.3.2.3 Image Optimization Stack (CDN)



Figure 6: Image Optimization Stack Diagram

The image optimization stack enables dynamic image processing and efficient content delivery through a Lambda@Edge-compatible setup using CloudFront, S3, and a custom image-processing Lambda function. The architecture ensures secure storage, fast delivery, and format-aware optimization of images based on request parameters.

Key Components and Their Roles

Amazon S3 Buckets

- **Original Image Bucket**
 - **Role:** Stores original uploaded images.
 - **Security:**
 - Public access blocked.
 - Encrypted with AES256.
 - **Automation:** Integrated with a Lambda-backed custom resource for auto-deleting stale objects.
- **Transformed Image Bucket**

- **Role:** Stores optimized and resized versions of original images.
- **Lifecycle:**
 - Automatically deletes objects after 90 days to save storage.
- **Access Control:**
 - Accessible to CloudFront via Origin Access Control (OAC).
 - Permissions granted to the image optimization Lambda for writing transformed images.

CloudFront Distribution

- **Role:** Delivers optimized images globally via `cdn.dripdropco.com`.
- **Configuration:**
 - **Aliases:** `cdn.dripdropco.com`.
 - **Viewer Protocol Policy:** `redirect-to-https`.
 - **Cache Policy:** Custom cache policy configured with long TTLs and no forwarded query strings.
 - **Response Headers:** Adds `x-aws-image-optimization` and `vary: accept` headers.
 - **Origins:**
 - **Primary Origin:** Transformed image bucket.
 - **Failover Origin:** Lambda Function URL (image processor).
 - **Functions:**
 - A **CloudFront Function** rewrites image requests to include processing parameters (format, width, height, quality).

Lambda Function (Image Processor)

- **Role:** Dynamically transforms and optimizes images based on query parameters.
- **Runtime:** Python 3.12 with a shared Lambda layer for image processing libraries.
- **Environment Variables:**
 - Buckets and TTLs are configured for optimization.
- **Access Control:**
 - IAM Role with permissions to read/write to both original and transformed S3 buckets.
- **Invocation:**
 - Invoked by CloudFront using a Lambda Function URL (IAM-authenticated).

AWS Certificate Manager

- **Role:** Manages SSL/TLS for `cdn.dripdropco.com`.
- **Validation:** DNS-based validation using Route 53.
- **Usage:** Applied to the CloudFront distribution for HTTPS support.

Route 53

- **Role:** Maps `cdn.dripdropco.com` to the CloudFront distribution.
- **Configuration:** A-type alias record points to the CloudFront domain using hosted zone.

Image Optimization Workflow

1. Request Handling

- A request for an image (e.g., /image.jpg?format=png&width=500) is sent to cdn.dripdropco.com.
- The **CloudFront Function** rewrites the URL path based on valid query parameters.

2. Cache Lookup

- CloudFront checks the cache for the transformed image.
- If not cached, it attempts to fetch from the **transformed image bucket**.

3. Lambda Invocation

- If the image isn't found in the transformed bucket, the request fails over to the **image optimization Lambda** via function URL.
- The Lambda reads the original image, applies transformations (resize, format, quality), stores the result, and returns the image.

4. Delivery

- The transformed image is cached in CloudFront and served with custom headers for future requests.

Security and Access Control

- **IAM Roles and Policies:**
 - Image processing Lambda has scoped permissions for S3 access.
- **Bucket Policies:**
 - Both buckets restrict public access.
 - Require SecureTransport (HTTPS).
 - CloudFront and Lambda are the only permitted principals.
- **CloudFront Origin Access Control (OAC):**
 - Used to securely connect to S3 and Lambda origins with SigV4 signing.
- **HTTPS Enforcement:**
 - TLSv1.2_2021 minimum via CloudFront.
 - ACM-issued certificate secures the CDN domain.

4.3.2.4 Image AI Processing Stack



Figure 7: Image AI Processing Stack

This architecture powers serverless AI-driven image processing using AWS Lambda, Step Functions, S3, SQS, and DynamoDB. It enables scalable and asynchronous segmentation, classification, and result aggregation of image data through a distributed workflow pipeline.

Key Components and Their Roles

Amazon S3

- **Role:** Stores processed image results.
- **Configuration:**
 - Bucket: ai-image-processing-results.
 - Used by Lambda functions for reading and writing segmentation/classification outputs.
 - Retention policies preserve data across deployments.

Amazon DynamoDB

- **Role:** Stores metadata and classification results for each processed image.
- **Configuration:**

- Table: ImageProcessingTable with image_id as the partition key.
- Provisioned throughput (5 RCU, 5 WCU) ensures predictable performance for metadata lookups and updates.

Amazon SQS

- **ImageProcessingQueue:**
 - **Role:** Triggers the processing workflow when a new image is ready.
 - Integrated with the Step Functions trigger Lambda.
- **ClassificationResultsQueue:**
 - **Role:** Receives the final classification results.
 - Used by downstream consumers to act on processed insights.

AWS Lambda Functions

- **SegmentLambda:**
 - Container-based function for image segmentation.
 - Reads input from S3 and writes intermediate results back.
- **ClassifyLambda[1–5]:**
 - Each handles classification using different AI models (e.g., classify1.pt, classify2.pt, etc.).
 - Invoked in parallel within the workflow.
- **MergeClassificationsLambda:**
 - Aggregates outputs from all classification models into a unified result.
- **SqsTriggerLambda:**
 - Listens to ImageProcessingQueue.
 - Initiates the Step Function execution upon new message arrival.

AWS Step Functions

- **Role:** Orchestrates the complete image processing pipeline.
- **Workflow:**
 - Starts with segmentation.
 - Runs multiple classification models in parallel.
 - Merges classification results.
 - Sends output to ClassificationResultsQueue.
- **Resilience:**
 - Configured with retries, exponential backoff, and graceful error handling for Lambda failures.

AI Workflow Overview

1. **Image Arrival:**
 - A new message lands in ImageProcessingQueue triggering SqsTriggerLambda.
2. **Segmentation:**
 - SegmentLambda processes the image and returns regions of interest.
3. **Parallel Classification:**

- Step Functions invoke ClassifyLambda[1–5] in parallel for different AI models.
- Each Lambda performs inference and stores results in DynamoDB and/or S3.
- 4. **Merging Results:**
 - MergeClassificationsLambda consolidates classification outputs.
- 5. **Result Delivery:**
 - Final result is sent to ClassificationResultsQueue for downstream processing.

Security and Access Control

- **IAM Roles & Policies:**
 - Each Lambda function is scoped to only required permissions (e.g., s3:GetObject, dynamodb:PutItem, states:StartExecution).
- **S3 Bucket Permissions:**
 - Lambda functions can only access their relevant S3 paths with write or read-only permissions.
- **Step Function Role:**
 - Allows invocation of only the specified Lambda functions.
- **SQS Permissions:**
 - Message visibility, deletion, and processing are restricted to the appropriate consumers.

4.3.3 Functionality

Overview: Our application, 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 and save outfits they like and cheaper options for a similar look with the help of AI. 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 created an intuitive and straightforward design that all users can easily understand, use, and navigate. We have a navigation bar at the bottom of our interface to allow for easy navigation across our main pages including home (feed), search, bookmarks, post, and profile. Along with that, many of our components are interconnected and easily link to each other. For example, clicking on a username from a feed post links to that user's profile page.

Feed: For users to find outfits they like, we have a feed, which is the first page shown upon entering the website or the application after login/signup. This feed displays 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 prefer and would like to see on their feed. If a user finds a post in their feed that they like, they can save or “bookmark” that post for future reference.

Product Suggestions: We have used AI to implement functionality to find cheaper options for a specific look. Using AI, our platform tags various products and then uses those tags to find similar products 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 generate an AI suggestion section that provides alternate items with a similar look.

4.3.4 Areas of Challenge

Infrastructure Implementation: One of our biggest challenges near the beginning of our project was implementing our infrastructure designs. A core objective of our project was to fully leverage AWS cloud tools for our entire infrastructure. However, this decision introduced a steep learning curve. Only half of our team had any prior experience with cloud technologies, while the remaining members were starting from scratch. To make meaningful progress on our project, we had to get the infrastructure in place, and this took us longer than expected. It required a lot of individual learning and effort to complete. Once we realized the importance of this challenge, we focused the majority of our effort there and got it done. In the end, it was very valuable for everyone on the team to gain a lot of hands-on experience working with AWS and creating a complete cloud solution.

Feature Prioritization: Another major hurdle was determining how to effectively prioritize features. Our initial design documents from Semester 1 were highly ambitious, envisioning a fully functional social media and shopping app. However, as development progressed, we quickly realized that not every feature could be completed we originally hoped for within our timeline. This led to numerous team discussions and debates about which features were critical for an MVP and which needed to be postponed or removed entirely. We addressed this by re-evaluating our requirements, ranking features by user impact and development effort, and establishing goals that focused on delivering the most value in the remaining time we had.

AI Recommendations: Aside from our complex cloud infrastructure, the other key component of our project was our AI model and AI recommendations feature. Creating and training our AI model was a challenge alone, but finding a good way to implement it into our app was by far one of our most complicated challenges. Backend wise, we had to figure out the most efficient way to pass user inputted data into the AI workflow pipeline, and find how to properly populate the database with the AI results. Then, we needed to utilize the AI data to provide similar clothing items to users. This required many discussions on our backend table design, and we eventually had 7 different tables to handle all the AI work. The key to overcoming this challenge was communication. In the second semester, a lot of our time during our weekly meetings was spent on this issue. By tackling this challenge together, we were able to make sure every step of the process was aligned, so that when we combined all of the process together, it integrated smoothly.

Communication and Collaboration: Although our team generally maintained a strong level of communication, coordinating across schedules and maintaining alignment during development phases proved to be a recurring challenge. With six members working on different aspects of

the project at once, it was sometimes difficult to keep everyone informed about all the changes that were happening. This issue showed itself in the final month of the project, as we began integrating all major components into a fully functioning application. During this phase, previously isolated features began to conflict, and several components failed to interact as expected. These errors revealed areas where our earlier communication had fallen short. To address this, we conducted a review of where we messed up, identified key points of failure, and refactored our code to restore cohesion. We also established better weekly meeting practices to make sure every update was being communicated effectively.

4.4 Technology Considerations

Cloud Provider: Amazon Web Services (AWS)

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.

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. Lastly, AWS's CDK gives it an edge over other providers. With the AWS CDK, the infrastructure can be written as code, allowing the code to manage and provision computing infrastructure, instead of relying on manual processes. This promotes automation, repeatability, and efficient management of resources.

Database: MySQL on Amazon Aurora

Relational vs Non Relational Database

A relational database (MySQL) was chosen over a non-relational database for several key reasons. Relational databases excel in handling structured data with well-defined relationships, which is crucial for a platform. For instance, users, posts, comments, and likes are all interconnected entities, and MySQL's ability to enforce relationships through foreign keys ensures data integrity and consistency. Additionally, relational databases support complex queries, such as retrieving all posts by a specific user or finding the most-liked posts, which are essential for social media functionality. MySQL's ACID (Atomicity, Consistency, Isolation, Durability) compliance guarantees reliable transactions, ensuring that operations like creating a post or updating user profiles are executed safely and reliably. While non-relational databases offer flexibility and scalability for unstructured data, drop's structured data model, need for complex queries, and emphasis on data integrity made MySQL the more suitable choice for our application.

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, 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, and the cost can be quite high. 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.

Frontend Technology: React

React vs. Angular

For our frontend 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.

For the front-end development, we chose React. 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.

5 Testing

Given the nature of our purely software focused project, testing has been something we have done since we started developing. Much of our testing has been similar to what a software engineer in industry would do, after implementing a change, we test that our change actually works. Our development style has been very agile, as we all make very frequent and small changes. This means that a lot of our testing is done by the individual who made the change, so it is not very structured.

Our project has come with a lot of work, as we have had to create a website and mobile app from scratch, all hosted and powered by the cloud. We set large goals for everything that we wanted to get done, and we've been able to do a lot of learning through doing. It's taken us pretty much up until the end of the semester to complete all of the features we wanted, so we haven't been able to devote much time to structured testing measures that you would see from a technology team at a large company.

Aside from routinely testing on our own, our team also implemented 3 more structured methods of testing: Integration Testing, Frontend Testing, and User Testing. Below we discuss those methods, as well as how we would address the other sections given more time/resources.

5.1 Integration Testing

To test our backend lambda functions and our API Gateway endpoints, we have created a suite of integration tests that call our endpoints and check that everything is working as expected. These tests are located in our backend folder in their own "test" folder. We are utilizing the python library **pytest** that allows us to call our API endpoints and assert that the backend lambdas return the expected data from our database. This allows us to test our core backend functionality in an efficient way, testing a variety of functionality at once: endpoint health, correct return format, lambda logic, data retrieval, and database status.

5.2 Frontend Testing

Frontend testing is essential for this React Native app to ensure UI components work correctly, user interactions behave as expected, and API integrations function properly. It prevents bugs during updates, validates critical user flows like photo posting, and maintains stability by mocking dependencies like cameras and permissions. Automated tests also safeguard against regressions, improving long-term maintainability. Ultimately, this leads to a more reliable, user-friendly app with fewer runtime errors.

The frontend tests were created using Jest, a widely-used JavaScript testing framework. These tests consist of component testing which verifies that components render correctly with their props and state. It also simulates user actions like button presses and text input, or interaction testing. Furthermore, the mocking of the context of the app and API calls were done to test component behavior under different states.

5.3 User Testing

Our project is all about users, so we knew that user testing would be important. When creating a complex mobile application, it's impossible to find all the bugs and errors without actually using the app. Additionally, we want our app to be easy and smooth to use, and we need to use the app to understand this aspect.

Once our app was becoming fully functional, our team designated team members to allocate time in their week to mess around on the app and record all bugs, errors, and observations. This really helped us find the critical changes that needed to be made.

Aside from our own team, we also used other people for user testing. Our faculty advisor was a big help. Once we showed him how to run our app, he was able to play around on it and find a lot of areas for valuable improvements. We also sought out some friends and roommates to provide additional user testing. Getting more eyes on it from non-technical backgrounds helped us see things that mattered to the everyday user.

5.4 Unit Testing

Backend Testing

The backend units involve all the lambda helper functions that access the database. This semester, we tested our backend through our integration tests described above. Given more time, we would have used standard unit testing by testing each function individually with controlled/mock inputs. It will be essential to test edge cases and invalid input. We would use Pytest, a testing framework for Python, to write these tests.

Frontend Testing

The frontend unit tests were created using Jest, a widely-used JavaScript testing framework. These tests consist of component testing which verifies that components render correctly with

their props and state. It also simulates user actions like button presses and text input, or interaction testing. Furthermore, the mocking of the context of the app and API calls were done to test component behavior under different states.

API Testing

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

5.5 Interface Testing

Interface Overview: The project's four primary interfaces are the React-based Frontend, API Gateway, lambda functions, and the database.

Testing Unit Composition: Three primary compositions exist to test all of the interfaces properly. Backend & Database, API and Backend, Frontend & API. As described, our interface tests already test the communication between the database, backend, and API endpoints. Formal testing was not implemented for the API and Frontend connection. 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.

5.6 System Testing

Above we have already discussed the unit tests and interface/integration tests that are required for this project. Our current testing lays a solid foundation for system testing, as the integration tests can tell us a lot about the health and status of our infrastructure and systems. To pass the tests, our infrastructure must be set up properly, communication between systems must be active, and our database must be returning properly.

Future steps: 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.

5.7 Regression Testing

Currently, we have no formal regression testing in place. AWS was helpful in the fact that when we deploy to AWS, many potential errors and bugs were caught by their system prior to deployment completion. We also used our integration tests as a way to check that things were still working as expected.

Future steps: To fully implement efficient regression testing, one of our next steps if this project continued would be to add our unit and integration tests to our gitlab pipeline. This would automatically run our test files so that all tests would have to pass before changes could be made. This would ensure that no old functionality would be broken by new changes. We would need to create scripts that would automatically run the test files.

5.8 Acceptance Testing

Our acceptance testing approach was practical and aligned with how real-world software teams validate deliverables with stakeholder expectations. Rather than building a formal acceptance testing framework, we conducted review sessions with our faculty advisor, who acted as our primary stakeholder.

In these sessions, we walked through each of our project requirements one by one, demonstrating how we had fulfilled them. For **frontend features**, we showcased the mobile application, highlighting navigation flow, interactive elements, and UI responsiveness. This allowed us to validate not just technical correctness, but also usability and user experience.

For **infrastructure-related requirements**, we leveraged AWS's built-in visualization tools to present our cloud architecture. This included services like API Gateway, Lambda, RDS, and S3, all of which we connected into a cohesive and scalable system. We also explained our CI/CD pipeline and how infrastructure components interacted.

On the **backend side**, we used MySQL Workbench to walk through our database schema and live data. We also showed the ER diagram we created to show how our data was structured and how different components of our system connected. We also demonstrated real-time backend functionality by calling live API endpoints and observing responses, linking frontend actions to backend logic.

5.9 Results

Our testing shows that dripdrop meets core functionality requirements while also providing an intuitive and enjoyable experience for any and all users. Throughout our development process, we used a CI/CD pipeline, and manual testing, to ensure that any newly implemented changes did not unexpectedly break old functionality. For example, by utilizing **pytest**, we were able to validate that the API endpoints worked on every new backend/infrastructure iteration to ensure that the frontend did not experience unexpected downtime due to losing functionality on the backend. Despite full testing not being implemented in all sections, we were able to implement enough automated and manual tests to ensure that we did not unexpectedly break past features with new implementations. Overall, we were able to test properly throughout our development period to ensure that we delivered on core requirements. Through user testing, we were able to confirm that all of our features and requirements were satisfied. User testing results showed that the current product is a reliable, scalable, user-friendly platform that achieves all of our primary goals.

6 Implementation

6.1 Frontend

6.1.1 Mobile Application

The mobile application is the core product that we produced, and it features all of our most up to date progress. In the following section, we explain the flow of a typical user, and all of the most important screens and features implemented.

Login & Signup

Upon opening the application, the user will be greeted with the log-in page where they may log-in, using their email and password, given that they already have an account. If they do not have an account, they will want to navigate to the sign-up page via the “Don’t have an account? Sign Up” button below the “Sign In” button.

Once navigated to the sign-up page, they are required to enter a unique username, a unique, valid email address, and a valid password. The password must be at least eight characters long and include one uppercase letter, one lowercase letter, one number, and one special character. Upon clicking sign-up, with the previous fields all being valid, an email is sent to them with a confirmation code which they must enter within 60 seconds. If this is done, they will be redirected to the log-in page with a new account. They may then log in.

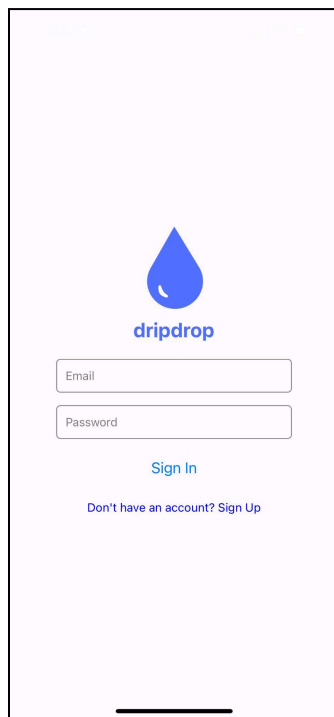
The login page features a light purple background. At the top center is a blue water drop icon with a white 'd' inside, and the word 'dripdrop' in blue text below it. Below the logo are two white input fields: 'Email' and 'Password'. Under these fields is a blue 'Sign In' button. At the bottom, there is a link that says 'Don't have an account? Sign Up' in blue text. A black horizontal bar is at the very bottom of the screen.

Figure 8: Mobile app login page

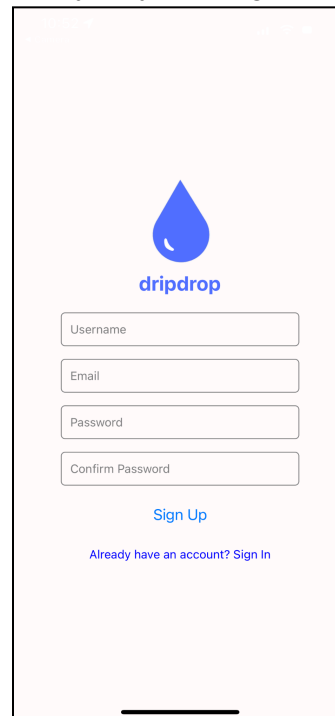
The sign up page has a light pink background. It features the same blue water drop logo and 'dripdrop' text at the top. Below the logo are four white input fields: 'Username', 'Email', 'Password', and 'Confirm Password'. Under these fields is a blue 'Sign Up' button. At the bottom, there is a link that says 'Already have an account? Sign In' in blue text. A black horizontal bar is at the very bottom of the screen.

Figure 9: Mobile App Sign up

Feed

Once the user has signed in, they will be redirected to the home/feed page. They will be shown the most relevant posts, based on date posted, whether they follow the user who created the post(s), if they have viewed the posts prior, etc. If there are no posts available for them at the time, they will be prompted with text and a button to “reset” the feed, which will reset the list of posts the user has seen. This “reset” feed is also shown once the user finishes scrolling through their present feed. The two possible screens upon logging in are shown by “Feed Page” and “Reset Feed Page”.



Figure 10: Mobile App Feed

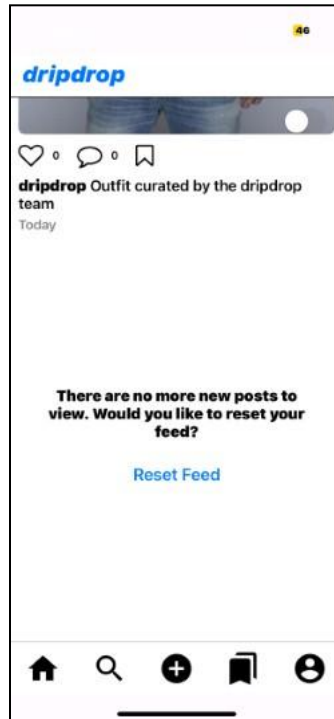


Figure 11: Reset Feed Message

From the feed, the user may like and/or comment on any given post in their feed at any time. They may also click on the user's name to navigate to their profile, which applies to both on the feed and in the comments section. The user may also click the toggle switch in the bottom-right of the post, which will be present if markers are present, which will show all of the markers for the clothing item that the AI found, and that details were provided for. The markers being visible may be seen below as "Feed Page - Markers Visible" and, in the same image, it can be seen that the post has been liked as well. In the "Comments Modal" image, the comments for the current post can be seen and the user may post new comments as well. Additionally, the user may "bookmark" a post by clicking the bookmark icon beneath the image. All bookmarked posts are displayed on the bookmarks page, as seen in *Figure 16*.



Figure 12: Feed - Markers Visible

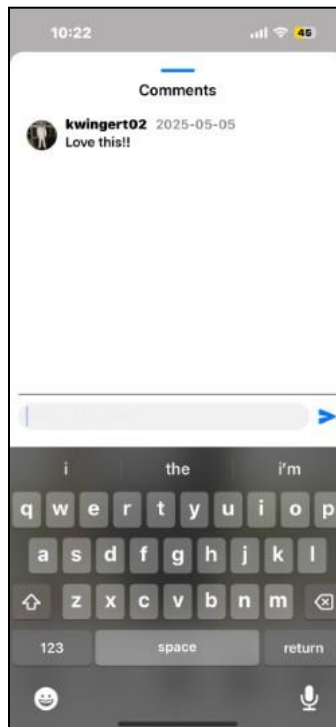


Figure 13: Post Comments

Upon clicking on a marker for a post, a pop-up will appear with all related clothing item information that the original poster provided. The pop-up is draggable to anywhere on the screen, at any time, and the currently selected marker will be highlighted in the background. Further, all markers for a given post may be cycled through by using the left/right arrows on the pop-up. If no arrows are present, then only one marker exists for that post. Importantly, the order remains the same no matter where the user begins in the list (i.e. it does not matter which marker is clicked first, the order remains). On the pop-up, the user may also click "Generate AI Suggestions" to see posts with similar items to the currently selected clothing item, as shown in *Figure 14*.

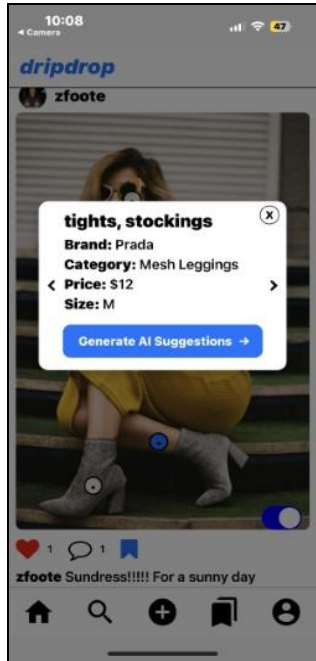


Figure 14: Item Information From Click

AI Suggestions

The AI Suggestions page, as seen in Figure 15, holds all of the posts that had a high match rate with the item the user wanted to view the similar items of. There is no limit on the number of posts that can match as a similar item, so there may be as few as one post or as many as all of the posts in the database. If the user clicks on one of the similar item posts, they will be taken to a feed of the matched posts, which they can scroll through.

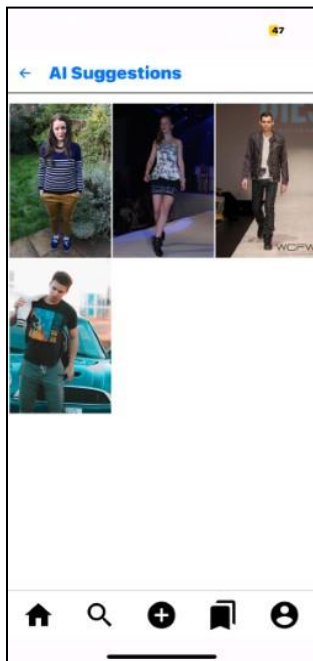


Figure 15: AI Suggestions Page

Bookmarks

The Bookmarks page, as seen in *Figure 16*, contains all of the posts that the user has bookmarked, essentially storing the posts in an easier location to be viewed at a later point in time. If a post within the bookmarks page is clicked on, the user is taken to a feed of bookmarked images that they may scroll through. The user may easily unbookmark the post by clicking the bookmark icon again. Once a post is unbookmarked, it will instantly disappear from the bookmarks page.



Figure 16: Bookmarks Page

Search

The search page, which is accessed from the magnifying glass in the bottom navigation bar, allows users to find other users and posts through a search query.

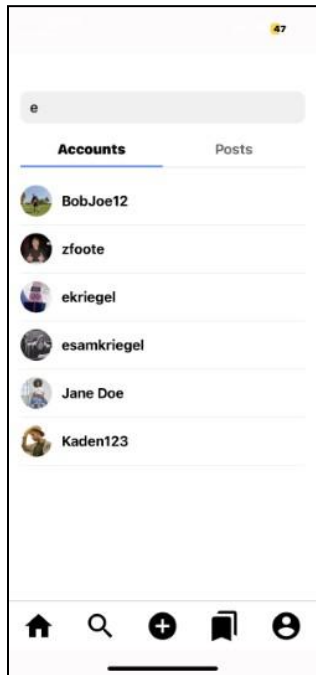


Figure 17: Searching Users



Figure 18: Searching Posts

When the user first opens this page, they are met with a search bar where they can enter their query. If they would like to search for users, they can enter the username or a portion of the username of the user they would like to find. Once they find the user they were looking for, they can tap that user's username and the app will redirect them to that user's profile page.

Alternatively, they could find specific posts by entering a phrase and pressing the "Posts" button and they will be able to see all posts with captions that contain their queried phrase. Tapping on a post shows that post in a feed containing all of the other search results. From here, the user can like and comment on that post

Create Post

Users are able to post images of their outfits for other users to see on their feeds and get inspiration from for their own collections. In order to create a post, the user must tap the plus button in the navigation bar at the bottom of the screen. This then brings the user to a screen where they can either take a photo through the app or select and upload a photo from their camera roll.

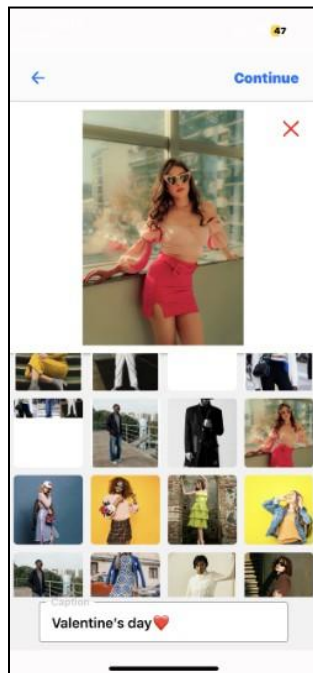


Figure 19: Image Selection Screen

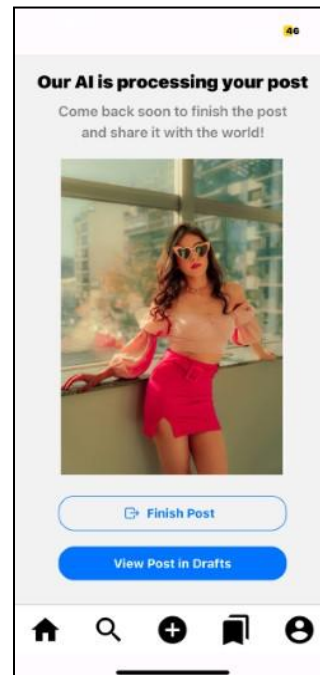



Figure 20: Image is Processing Screen

Once a photo has been selected, the user can write a caption for their post and select “Continue” at the top right hand corner of the screen. The photo is then sent to the backend for AI processing. They are then brought to a screen confirming that the post is being processed.

From here, the users can either wait on this screen until the AI is completed and the “Finish Post” button allows them to proceed, or they can navigate to the “Drafts” section on the profile page, where the post will appear once it is done being processed. The user can go to the “Preview Your Post” screen by simply clicking on the post in drafts.



The 'Verify Clothing Item Details' screen features a series of input fields for item information. The fields are: 'Item Name' (containing 'top, t-shirt, sweatshirt'), 'Brand', 'Category', 'Price' (with a '\$' symbol and '0'), 'Item URL', and 'Size'. At the bottom, there are 'Back' and 'Save' buttons. The screen is part of a mobile app with a bottom navigation bar containing icons for home, search, add, bookmark, and profile.

Figure 21: Item Information Screen

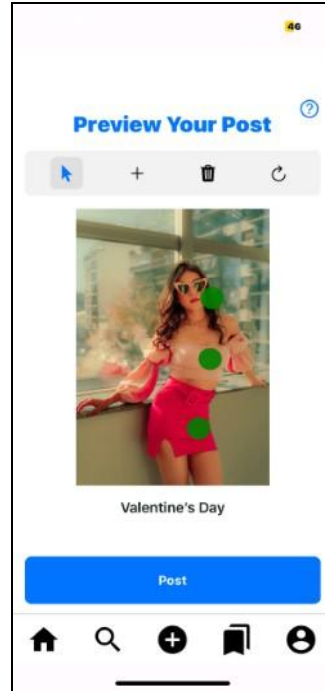


Figure 22: Preview Post Screen

When the user gets to the finish post screen, the image will already have green markers on some of the clothing items generated from the AI. The markers will have a type entered for them (e.g. t-shirt, dress, etc), but no other information. The user can add to this information using the inputs in Figure 21, choose to leave the marker alone and add no additional information, or even delete the marker. Additionally, the user can add additional markers as needed. After the user is done inserting clothing item information, they can click the “Post” button to officially make the post available to the public. The post will not show up in the “Posts” section of their user profile and can be seen in other user's feeds.

Profile Page

The user can navigate to their own profile page using the profile icon on the right side of the navigation bar. On their own page, the user may see two different tabs for posts, “Posts” and “Drafts”, where each will have unique posts, assuming the user has created posts. The “Posts” tab will show the currently public posts, so ones that everyone else on the platform can see. The “Drafts” tab will show all posts ready to be reviewed. A post is sent to “Drafts” once the AI has finished processing it and the user needs to confirm that AI processing was correct and/or add more information of their own. For both of these tabs, when an image is clicked, a “feed” is provided. As the user switches between the different tabs, the posts count at the top will update accordingly. See the “Profile Page - Posts” and “Profile Page - Drafts” images.



Figure 23: Profile Page - Posts

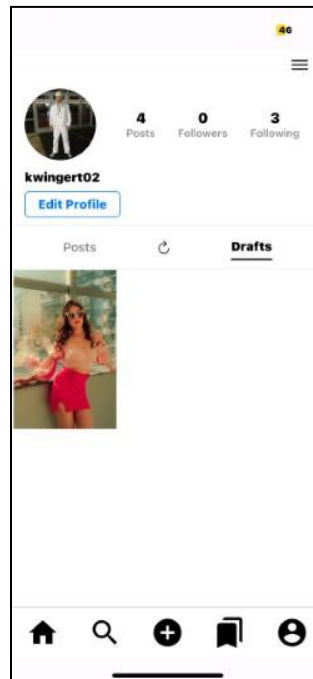


Figure 24: Profile Page - Drafts

While still on their own profile page, the user can click “Edit Profile” to edit their own profile where they may change their profile picture, username, email, and/or password. This is seen in the “Edit Profile Settings Page” image. In the top right, the user may also click the three horizontal lines to access a screen where they may also access the edit profile page or they may sign out. This is shown in the “Settings Page” image.

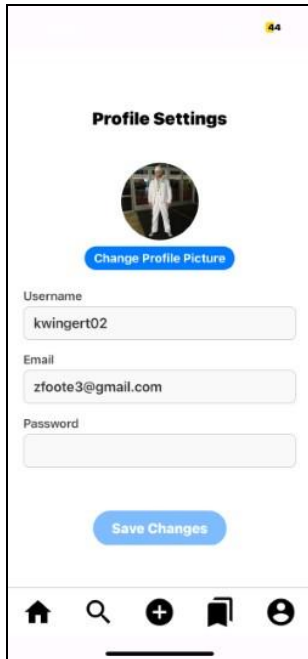


Figure 25: Edit Profile Settings Page

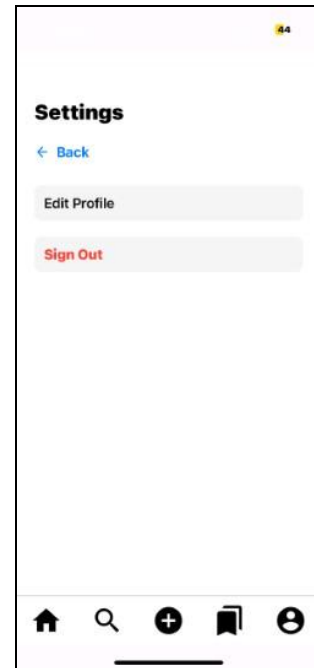


Figure 26: Settings Page

The user may also navigate to another user's profile page to see their public posts, follow/unfollow the person, and the number of posts, followers, and following they have. If the user clicks on any of the posts in the grid, they may then scroll through a "feed" like result to view all of that person's posts. One may also view that user's followers and following, i.e. the actual users. This can be seen in *Figure 27*.

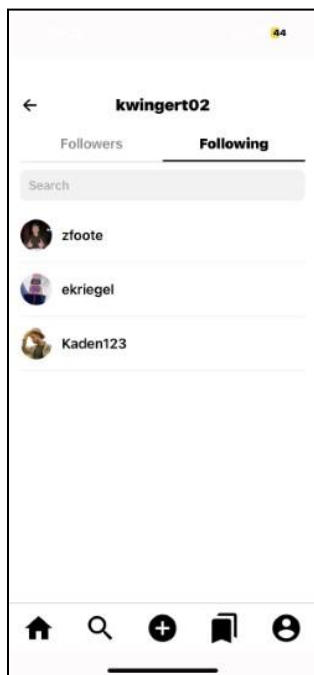


Figure 27: Following Page

6.1.2 Website

In the first semester, we worked on our application's web platform and the backend database to support it. The website served as our pilot frontend, and starting in the second semester, we completely switched focus to the mobile application. The website helped us get a MVP and allowed us to initially test backend functionality and infrastructure design.

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.

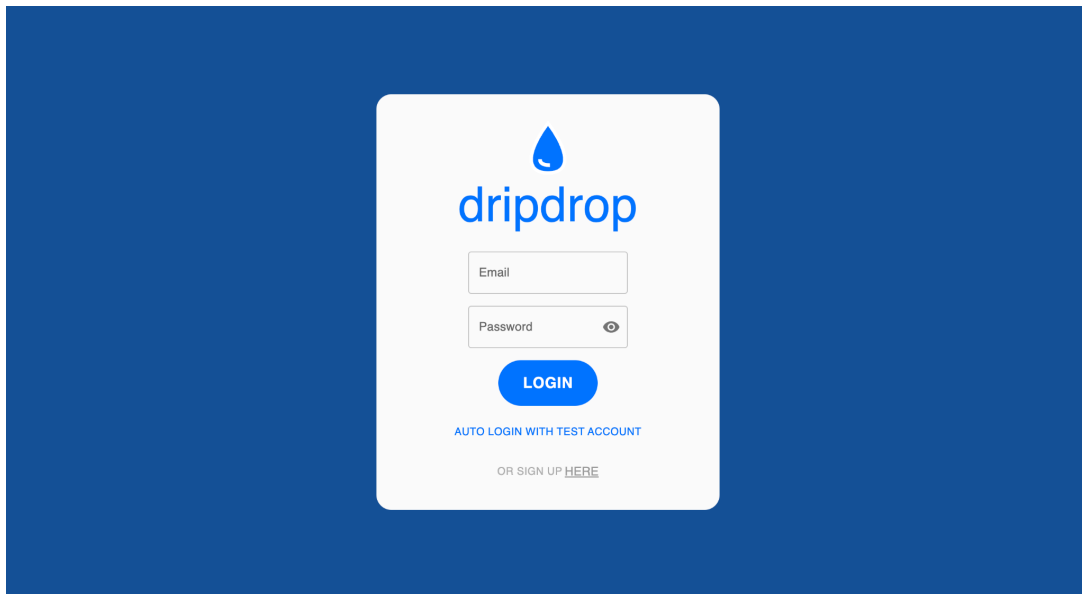


Figure 28: Website Login Page

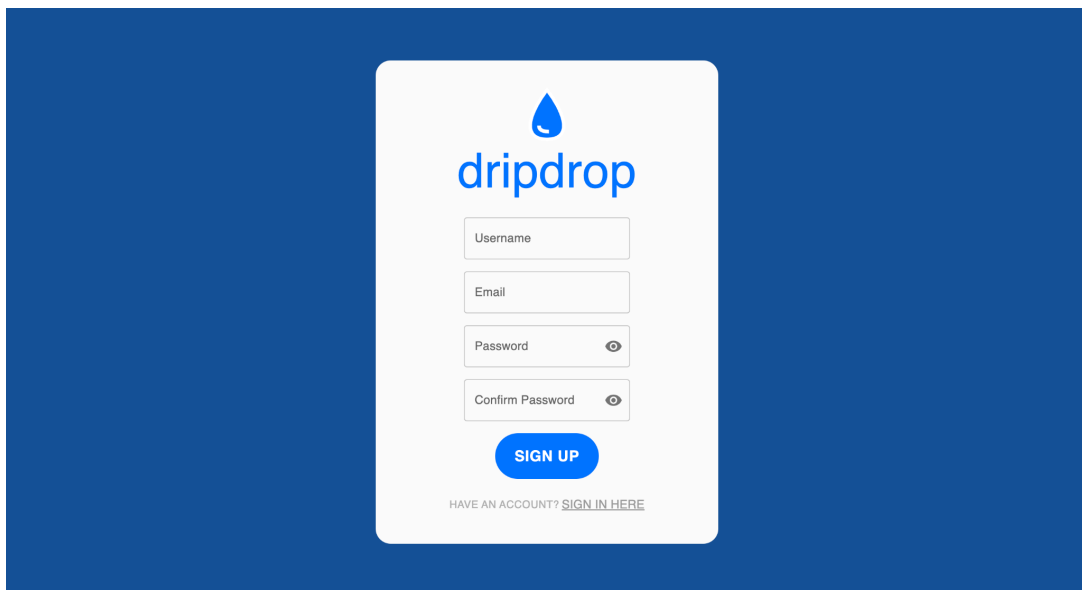


Figure 29: Website sign up page

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.

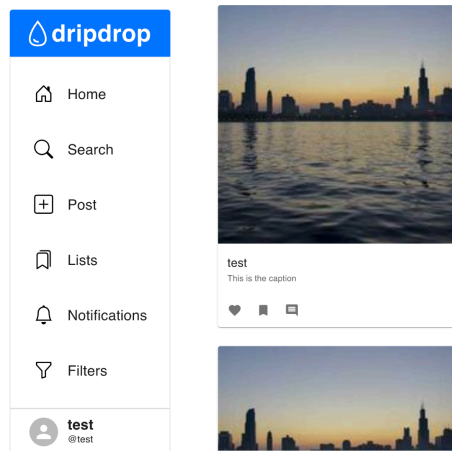


Figure 30: Website home page

We currently have early implementations of the home page, search bar, and posting page.

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.

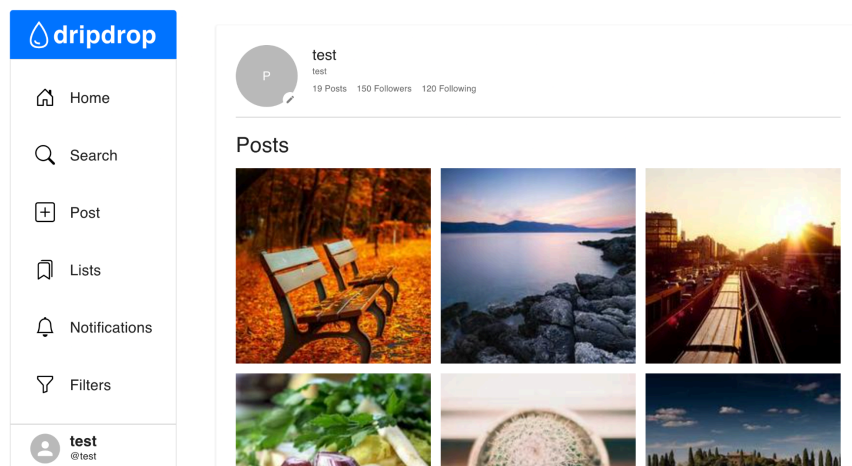


Figure 31: Website user profile page

6.2 Backend

We accomplished many of the backend features and functionalities that we set out to do last semester. We now have a total of 39 API Gateway endpoints that are handled by individual lambda functions. We have 57 total lambda functions, with some lambdas providing only backend functionality. Below is a summary of our API and lambdas from the AWS Console:

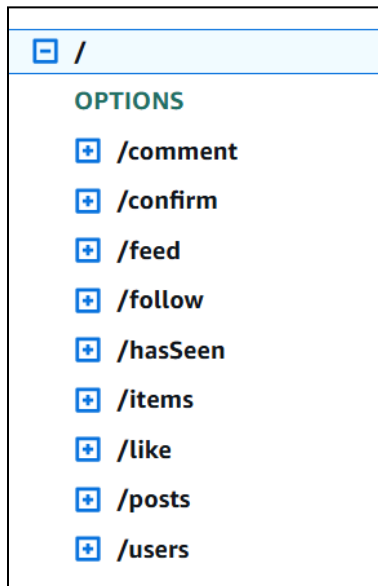


Figure 32: API Gateway endpoints

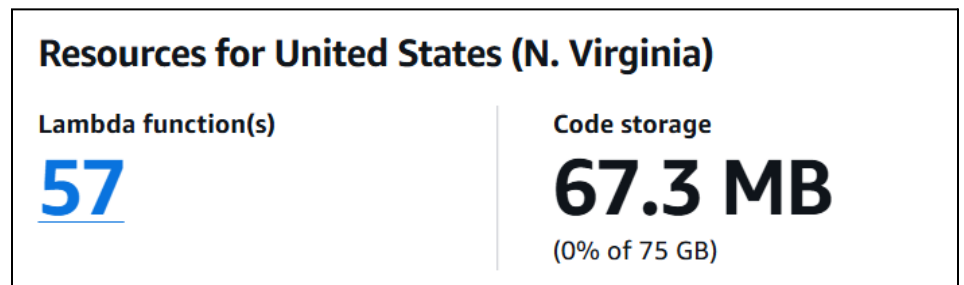


Figure 33: Lambda function count

As you can see above, we have API endpoints to handle a variety of frontend functionality. We are still using the same MySQL database from the previous semester, still hosted on AWS by Amazon Aurora. The only major changes to our backend structure was implementing a variety of security measures that we outline in section 6.4: Security. Below is our finalized ER Diagram:

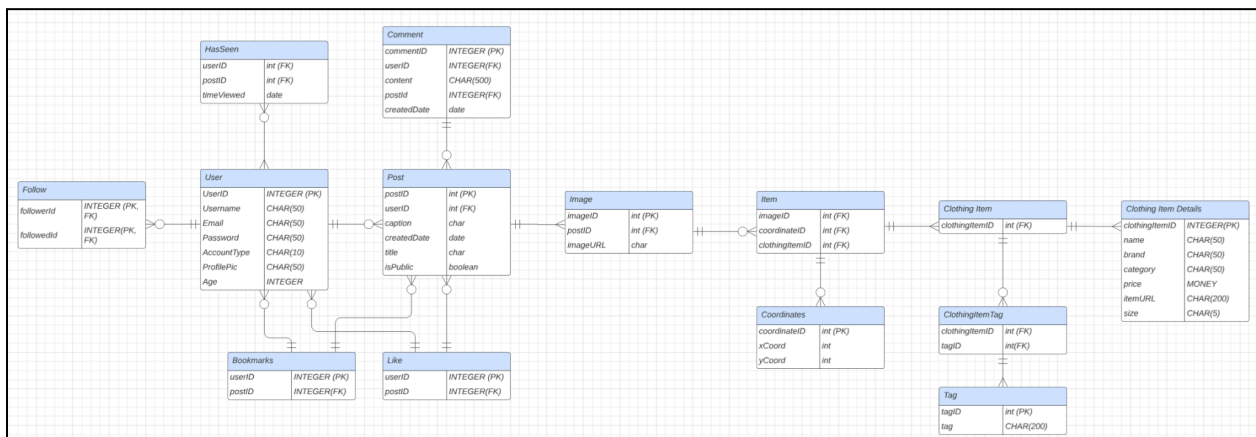


Figure 34: ER Diagram

Overall, our backend development was successful and played a critical role in enabling full functionality across our mobile application. By the end of the second semester, we had built a clean, well-structured, and scalable backend that supported all required features for the frontend. A key factor in our success was the consistent use of modular code patterns and shared helper functions. This allowed us to rapidly develop and deploy new API endpoints and database tables without unnecessary complexity. Additionally, the simple and uniform structure of our Lambda code made integrating Amazon Cognito for user authentication and security both straightforward and efficient. We were able to implement secure access controls and session management without disrupting our development flow.

6.3 AI Model

The backend architecture is designed as a serverless image analysis pipeline that leverages AWS Lambda functions, Step Functions, S3, and DynamoDB. The flow begins when an image is submitted for analysis—this image is fetched from a CDN and passed to a segmentation Lambda function, which uses a YOLO-based deep learning model to detect clothing items and segment them. The segmentation output includes cropped images, bounding boxes, and key attributes like color and category, which are then stored in an S3 bucket. A corresponding metadata entry linking the image to its S3 result is stored in DynamoDB for downstream access.

Once segmentation is completed, the classification Lambda functions are triggered. It retrieves the segmentation result from S3 using the image ID, then processes each cropped clothing item through another YOLO model trained to classify fine-grained attributes like style or type. This classification is done in batches for performance, and predictions are appended as attributes to each item. The final data structure—containing clothing labels, attributes, and coordinates—is returned, ready to be persisted in the database.

The results from classification are forwarded to a third Lambda function via an SQS queue, which acts as a buffer and enables decoupled asynchronous processing. This Lambda parses the results, extracts useful metadata (e.g., coordinates, clothing types, attributes), and maps them into the backend's relational schema. SQLAlchemy ORM is used to handle complex relationships between Image, ClothingItem, Tag, Coordinate, and Item tables. Each tag is either fetched or created on-the-fly to ensure deduplication, and coordinates are averaged to determine item placement. This structure enables rich querying capabilities for future features like search or recommendation.

To support state orchestration and ensure robust execution of this pipeline, AWS Step Functions manage the coordination between segmentation, classification, and database updates. If any step fails (e.g., missing S3 data or model inference errors), meaningful error messages are logged and optionally routed to fallback handlers like Dead Letter Queues (DLQs) for retry or inspection. This architecture improves reliability and observability while avoiding tight coupling between components.

The backend design focuses on modularity, scalability, and resilience. Deep learning models handle complex visual tasks, serverless functions enable elastic compute without infrastructure

overhead, and AWS-native services like S3, DynamoDB, and Step Functions glue everything together. This results in an efficient, event-driven pipeline that can scale to handle large volumes of image data while remaining cost-effective and easy to maintain.

6.4 Security

Frontend Security:

Our team put in place multiple security measures to ensure the users of the app are old enough to use the app, and that only the users within the app can make API calls, so that the app data is secure.

We implemented a calendar date selector for the user when the user is creating their account, to ensure that every user is at least 13 years of age, which is the minimum age in the U.S. for an app to allow a user to use a social media application. Additionally, when the user enters all of their account information and their birthday, they are instructed to perform a two-factor authentication by retrieving a confirmation code that we send to their email. This ensures that users are validly entering their own email.

The largest part of our fronted security is that upon sign in, the `/signin` endpoint returns an `id_token`, an `access_token`, and a `refresh_token`. All tokens are JWT encrypted, so on the frontend we decrypt the `id_token` to retrieve user information (uuid, email, expiration time for the token, etc.) and then we store it in the session. Then, we insert the `id_token` as a bearer token in the header of every API call, which allows us to get information from the backend, which requires the bearer token. When the `id_token` expires, we can call the `/refresh` endpoint and insert the refresh token to get a new `id_token` and `access_token`. The `access_token` determines the level of access that the user has within the app, but currently all users have the same amount of access so this is not used yet.

Backend Security:

Backend security in this system is primarily handled through AWS Cognito, which manages user authentication and verification. During registration, users provide details like email, username, password, and date of birth. These credentials are validated and securely stored in the Cognito User Pool, which ensures password policies and email verification flows are enforced. Cognito responds with a confirmation code sent to the user's email, which the user must submit to confirm their identity. This ensures that fake or unverified accounts can't proceed further in the system.

Once a user confirms their identity using the correct code, Cognito's `admin_get_user` is used to fetch the user's UUID (sub), a globally unique identifier. This UUID, along with the user's profile data, is then sent to an internal Lambda function responsible for writing this information to a backend database. If this database operation fails, the system proactively rolls back the Cognito user to avoid dangling accounts. This rollback mechanism adds a layer of integrity and security, ensuring the user is only fully registered if all backend services succeed.

For authentication during login, Cognito's `initiate_auth` method is used, enabling secure, token-based sign-in. Upon successful authentication, the system returns an ID token, access token, and refresh token to the user. These tokens allow authenticated access to protected backend routes without exposing raw credentials. Any failure in authentication—such as incorrect credentials or expired codes—is handled securely, with error messages returned and no sensitive information leaked. This architecture not only keeps user data safe but also ensures secure, scalable identity management throughout the application lifecycle.

6.5 Reflection - What worked and what didn't

What worked:

Infrastructure: Our infrastructure is one of the highlights of our project. We have fully implemented the infrastructure we designed last semester on AWS. Our infrastructure handles our database, backend code, API endpoints, frontend code, security, image storage, and more all using AWS and industry standards.

Backend/API: Our backend development has also been very successful. Our backend code is clean and consistent, and handles a wide variety of frontend functionality needed to have a working app. We implemented almost all of the tables and endpoints that we initially planned last semester.

App: Our frontend development this semester has been almost solely focused on our iOS/Android app, and we are very pleased with the progress we have made. The app has good usability, with many features acting as expected for a social media app.

AI Model: The AI model had the most uncertainty going into this semester on whether we would be able to complete it. We are proud to say that we have completed a full AI workflow that allows users to get AI recommendations for outfits they like. The process of training a model and integrating it into our backend and frontend design gave us a lot of valuable learning opportunities.

What does not work as expected:

Website: Last semester, we created a website acting as our prototype through the initial stages of planning and development. Starting this semester, we focused all of our resources on the mobile application and have since made many changes from the initial website design. As a result, the website did not have many updates this semester, and is lacking a lot that the mobile app offers.

Endpoint Latency: The latency of our API Gateway endpoints are slower than desired, and this causes long loading times on our mobile application. We explored numerous fixes throughout

the semester, but the solutions required much higher monthly costs that were outside our budget.

Affiliate links and DripDrop rewards: In the first semester, during planning, we discussed ways this app could generate money through affiliate links and dripdrop rewards based on users interacting with your posts. In reality, this was something that was not as crucial to the core functionality of the app, and we did not have time to address this initial design plan.

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.
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 receive 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."	Everyone is given a chance to speak at the 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."	We will focused on protecting user data through our security measures. This includes ensuring only people above the age of 13 are allowed to use the app, and requiring bearing tokens when making API requests.
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	We docuent proper citations to credit the creators when borrowed work is used.
Sustainability	Designing the app to be efficient, minimizing resource usage, and supporting long-term maintenance.	PUBLIC: "Act consistently with the public interest."	We worked to improve the speed and efficiency of the app though removing the unnecessary API calls and caching data.
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 strict security measures including requiring an authentication token to make backend requests.

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.

An area where we can improve: Work Competence

Our team focused on learning and growing our skills. To improve our process and software development lifecycle, we should implement more rigorous quality assurance processes, such as peer code reviews before merging code into the master branch.

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 receive 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	Users can receive money from companies if another user purchases a product using their referral link

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 are 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 work 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, I 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 reasoning, 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 Summary Of Progress

Summarizing what we have done so far, our team currently has a working AWS Aurora server hosting our database and website. We have a functional backend, including all of the needed lambdas for user functionality. Our team also has a mobile frontend that allows users to view their feeds, save and create posts, and search user profiles. Additionally, the team has a working AI model that can tag and differentiate attributes of apparel items in an image. This AI model is also capable of providing suggestions for posts and articles of clothing with a similar look to other potentially more expensive articles.

We have met our initial goal to have a working website with the basic functionalities by the end of the first semester. We were also able to build a working mobile application along with a working AI outfit-matching model during the second semester. We also felt it was important to utilize and build out basic security measures to keep our backend database secure and safe. As a result, we placed a heavy emphasis on requiring authentication tokens to access all of our endpoints. Our main constraints through the development process have been finding the time between all the other commitments we've had as college students to develop an entire application. Additionally, there were a lot of new concepts for our team to learn which caused a slower development process. However, we were able to overcome these constraints and challenges to finish our project in time.

8.2 Value Provided

dripdrop is designed to meet the needs of users who want a simple, engaging, and visually appealing platform to share and discover fashion and apparel content. Accessibility is an important nonfunctional requirement of ours, which is why we designed *dripdrop* to be accessible through the web, iOS, and Android platforms.

Additionally, this project addresses the need for users to have their content seen by a wide audience. Users can create public posts which cater to those who want to share their ideas and creativity with the world. *dripdrop* was designed to solve specific problems in the social media space such as overwhelming complexity and being cluttered with unnecessary features. That is why *dripdrop* focuses on simplicity, allowing users to share and consume content without distractions or annoying advertisements.

dripdrop fits into the broader context of social media by offering a modern, minimalist alternative to existing platforms. It aligns with current trends such as community building. *dripdrop* fosters a sense of community by enabling users to connect over a shared interest in fashion.

Some examples of the value *dripdrop* provides is for content creators. A photographer could use this app to showcase their portfolio which would allow them to gain exposure and potential clients. Additionally, small businesses can promote their products through public posts, reaching a broader audience without breaking the bank with excessive advertising. Lastly, everyday users can benefit from viewing others' posts through gaining inspiration for new clothes to buy.

While we have not released this app to the public yet, we have shown it to classmates who had a general response stating that they would use this app if it were available. They mentioned that being able to easily purchase the clothes that they see from social media posts would likely increase the likelihood of them making a decision to buy something they otherwise might not have.

8.3 Next Steps

1. Beta Testing with a Small Group of Users

We would like to gather feedback from a small group of beta users to identify bugs, usability issues, and areas for improvement. This involves recruiting a diverse group of beta testers, including content creators, everyday users, and businesses. We would provide clear instructions and feedback channels (e.g., surveys, in-app feedback forms) to gather their comments and suggestions. We would also monitor usage patterns and engagement metrics to identify pain points and successes. The goal of this beta testing is to use the feedback to refine the app's design, functionality, and performance before a full release.

2. Increased Testing Efforts

Due to the limited amount of time in this course, we felt that prioritizing the development of new features was more important than developing a comprehensive set of tests. However, we recognize the importance of having a thoroughly tested application, and with more time, the next steps would involve prioritizing testing alongside feature development. Our test plan includes implementing unit tests, integration tests, and end-to-end tests for both the web and mobile apps. To perform these tests, we would use testing frameworks like Jest (for JavaScript/TypeScript), Detox (for React Native), and pytest (for Python). Additionally, we would conduct regular user acceptance testing (UAT) to ensure new features meet user expectations. The goal of these tests are to create a more robust and bug-free application that delivers a seamless user experience.

3. Improving AI Autofill and Clothing Item Segmentation

The objective of improving AI autofill and clothing item segmentation is to enhance the accuracy and usability of these AI-powered features. Currently, these tools help users automatically fill in clothing information and segment clothing items in images, but there is room for improvement. To achieve this, we plan to collect more training data, focusing on a diverse range of clothing items and styles to ensure the AI models can handle various scenarios. Additionally, we will fine-tune the models using real-world data and feedback from beta testers to make them more robust and accurate. We will also explore advanced computer vision techniques, such as Mask R-CNN and YOLO, to improve the precision of clothing item segmentation. The outcome of these efforts will be a more reliable and user-friendly AI system that saves users time and enhances the quality of their posts by providing accurate and relevant information.

4. Developing a Marketplace Feature

The objective of developing a marketplace feature is to transform dripdrop into a trusted community platform where users can buy and sell clothing items. This feature will add significant value to the platform by creating a new revenue stream and fostering a sense of community. To achieve this, we will add a dedicated marketplace section where users can list items for sale, complete with detailed descriptions, photos, and prices. Secure payment processing and buyer/seller verification will be implemented to build trust and ensure safe transactions. Additional features, such as reviews, ratings, and messaging, will be included to facilitate communication between buyers and sellers. The outcome will be a vibrant and trusted marketplace where users can buy and sell clothing in a safe and community-driven environment, further enhancing DripDrop's value proposition and user engagement.

9 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.

Pintrest

Pinterest, Inc., "Pinterest: Visual Discovery Engine," 2023. [Online]. Available: <https://www.pinterest.com>

LTK

RewardStyle, "LIKEtoKNOW.it: Shop Influencer Outfits," 2023. [Online]. Available: <https://www.liketoknow.it>

Amazon Outfit Builder

Amazon, "Outfit Builder," 2023. [Online]. Available: <https://www.amazon.com>

Appendices

Appendix 1 - Operation Manual

Backend setup instructions and documentation:

<https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/Backend>

Website Frontend setup instructions and

documentation: <https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/Website-Frontend>

Mobile App Frontend setup instructions and documentation:

<https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/mobile-frontend>

API documentation and developer instructions:

<https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/API>

Git Hooks documentation:

<https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/Git-Hooks:-Pre-Commit-and-Pre-Push>

Lambda Design documentation:

<https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/Optimizing-AWS-Lambda-Design>

User Types documentation: <https://git.ece.iastate.edu/sd/sdmay25-25/-/wikis/User-Types>

Appendix 2 - Alternative/Initial Version of Design

For *DripDrop*, our initial approach evolved significantly as we gained more insight into the market and user needs. Below is a breakdown of the key design versions that were considered before settling on our current approach:

1. Initial Version: Chrome Extension for Finding the Best Deals

- **Description:** Our first version focused on creating a Chrome extension that would help users find the best deals on products across different e-commerce platforms.
- **Why it was Scrapped:** After conducting thorough market research, we realized that this idea had already been implemented in many existing tools. The concept was not unique enough to stand out in the competitive landscape of deal-finding extensions, so we decided to pivot and explore a different direction.

2. Shift to Mobile App Development

- **Description:** After evaluating user behavior and feedback, we realized that the majority of our target audience was more likely to interact with the app on mobile devices rather than through a browser extension or website.
- **Why the Pivot Happened:** The mobile app offered greater flexibility, personalization, and accessibility for users, aligning better with current trends in mobile commerce. This shift led us to prioritize the mobile app experience, and as a result, the website version of the app was put on hold for future development before it was fully completed.

3. Removal of Affiliate Link Feature

- **Description:** In our first semester, a core feature we wanted for our app was the ability for users to make money when other users used their links to purchase clothing. This requires partnerships with clothing companies, as well as a rewards system.
- **Why it was scrapped:** Early on in the second semester, we understood this was a good long term vision, but would not be possible in a semester. We wanted to focus on getting our app created, and implementing an AI recommendations feature. As a result, we cut all work on affiliate links and sponsorships.

Appendix 3 - Code

GitHub repository link: <https://git.ece.iastate.edu/sd/sdmay25-25>

Appendix 4 - Team Contract

Team Members

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

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.

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

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.

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

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 2:00 - 3:00 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:
 - Mobile application that offers all our desired features
 - AI recommendation feature complete
 - Minimum presence of bugs and errors
 - Incremental progress from each teammate (consistent weekly updates and progress).
 - All backend lambdas and endpoints needed for app complete
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 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