Atlanta Sustainable Fashion Week (ATLSFW) Mobile Application Enhancement

JID 4347

Team Members:
Kevin Tang, Ruixuan Gao, Siddhant Agarwal, Vivek Bumb, Xilu Zeng

Client:
Tanjuria Willis

Repository:
https://github.com/KTang603/JID-4347-ATLSFW-App-Enhancement.git

Table of Contents

List of Figures

Terminology

Actions: Redux functions that dispatch state changes (e.g., loginAction, saveAction)

AdminProfile: Administrator account interface

API: Application Programming Interface; the set of routes and endpoints in the server that handle client requests for authentication, articles, and user data

ATLSFW: Atlanta Software Framework; the name of this project platform for article management and user interactions

Components: Reusable UI elements in React Native that make up the application interface (e.g., Article, NavBar)

DataManagers: Server-side modules that handle data operations (e.g., articleManager)

decryptionUtils: Handles data decryption

Docker: Used for containerization as indicated by Dockerfile and .dockerignore

encryptionUtils: Handles data encryption

Environment Variables: Configuration values stored in environment_variables.mjs for secure application setup

format: Handles data formatting

hashingUtils: Manages password hashing and verification

JSON: JavaScript Object Notation; the data format used for client-server communication

JWT: JSON Web Token; used for secure authentication and authorization in the application

MongoDB: A NoSQL database used to store articles, user information, and other application data

Node.js: The server-side runtime environment used to run the backend server

PKI: Public Key Infrastructure; referenced in pki.mjs for security implementations

Profile Types: (Profiles listed separately as AdminProfile, UserProfile, VendorProfile)

React Native: A framework for building native mobile applications using React

Reducers: Redux functions that specify how the application's state changes in response to actions

REST: Representational State Transfer; the architectural style used for the API design

Routes: Server endpoints that handle specific types of requests:

- login: Authentication endpoints
- posts: Article management endpoints
- signup: User registration endpoints
- user: User profile management endpoints
- vendor: Vendor-specific operations

Redux: A state management library used to handle application state and data flow

Screens: Full-page components that represent different views in the application (e.g., LoginScreen, ProfilePage)

UserProfile: Regular user account interface

Utils: Utility functions for common operations:

- encryptionUtils: Handles data encryption
- decryptionUtils: Handles data decryption
- hashingUtils: Manages password hashing and verification
- format: Handles data formatting

VendorProfile: Vendor account interface

Introduction

The Atlanta Sustainable Fashion Week (ATLSFW) Mobile Application project is an initiative aimed at applying mobile technology to foster and propagate sustainable fashion practices. This application serves as a dynamic platform where fashion enthusiasts, vendors, and the general public can engage with a wealth of content centered around sustainable fashion. The application not only aims to inform users about sustainable practices but also facilitates active participation through interactive features that allow for content creation, management, and personalized user experiences.

Background

Fashion remains a largely unsustainable industry, heavily contributing to resource consumption and pollution. The Atlanta Sustainable Fashion Week is an event aiming to tackle these issues by promoting alternative and sustainable fashion. The ATLSFW Mobile Application is designed to tap into the potential of app based interaction by providing users with access to educational articles and a community-driven platform to discuss and promote sustainability in fashion. The application leverages advanced mobile technology to create a seamless and engaging user experience that encourages learning, interaction, and collaboration on sustainable fashion topics.

Project Scope and Objectives

The project is structured to deliver a comprehensive mobile application with robust features that support:

- **Content Management:**
  - Allowing users, especially vendors, to create, manage, and share articles related to sustainable fashion.
  - Supporting diverse media content, ensuring that users can share engaging and informative content.
  - Providing analytics on content performance (most like & saved articles)
  - Enabling content filtering and categorization via tags.
- **Enhanced User Experience:**
  - Intuitive and responsive design with seamless navigation between sections.
  - Personalized content feeds based on user interests and interactions.
  - Support for offline access to ensure users can interact with content without an internet connection.
  - Interactive calendar for event planning and participation.
- **Community Engagement:**
  - Event management system for promoting sustainable fashion events.
  - Workshop registration and participation tracking.
  - User interest tracking for events and workshops.
  - Vendor showcasing and direct shop access.

- **Role-Based Access Control:**
  - Differentiated experiences for regular users, vendors, and administrators.
  - Administrative tools for user management and content moderation.
  - Vendor-specific features for shop management and promotion.
  - User profile customization and preference management.
- **Security and Performance:**
  - Secure user authentication and authorization.
  - Protection of sensitive user data.
  - Performance optimizations for responsive user experience.
  - Scalable architecture to accommodate growing user base and content volume.

Target Audience

The application serves three primary user types:

1. **Regular Users:** Fashion enthusiasts and consumers interested in sustainable practices who use the app to discover content, events, and sustainable brands.
2. **Vendors:** Sustainable fashion brands, creators, and designers who use the platform to promote their products, share educational content, and connect with potential customers.
3. **Administrators:** Platform managers who oversee content quality, user management, and overall system integrity

Each user type has access to specific features tailored to their needs, creating a comprehensive ecosystem that supports the sustainable fashion community.

System Architecture

Introduction

The ATLSFW mobile application follows a structured architecture designed to support scalability, maintainability, and performance. This section outlines both the static and dynamic aspects of the system architecture, providing a comprehensive view of how different components interact to deliver the application's functionality.

Static Architecture

Diagram:

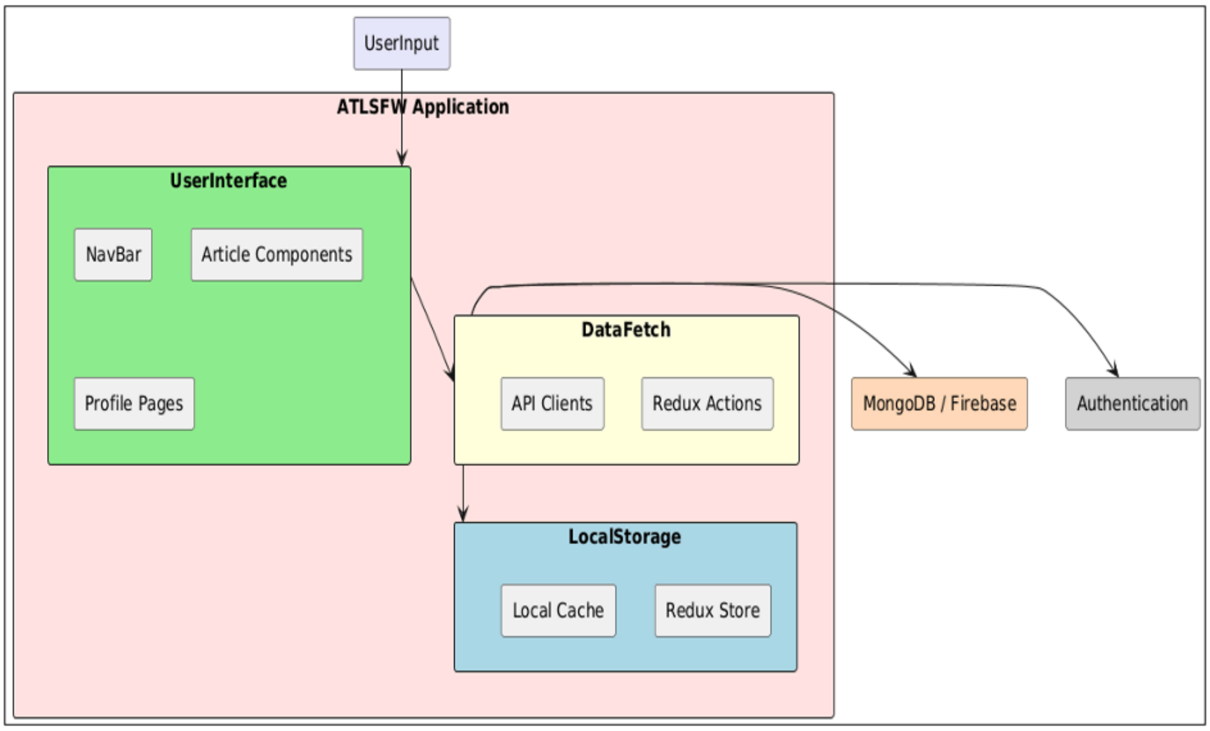


Figure 1.1: Static Architecture

Description:
The static system architecture illustrates the structural organization of the ATLSFW application, showing how different components are connected and interact with each other. This architecture was chosen to address several key aspects of the system's design and implementation.

Component isolation is a fundamental principle of the architecture. The design deliberately separates user interface components from data management functions, ensuring a clean separation of concerns. Clear boundaries are maintained between different functional areas, allowing each component to operate independently. This isolation enables independent testing and maintenance of components, making it easier to verify and update individual parts of the system. Furthermore, this approach facilitates future updates and modifications by minimizing dependencies between components.

The frontend layer is built using React Native, providing a cross-platform mobile experience with native performance. Key components include:

1. **UI Components**: Reusable interface elements like buttons, cards, and modals that maintain consistent styling and behavior throughout the application.
2. **Screens**: Full-page components representing different views in the application, such as HomeScreen, EventsScreen, and ProfilePage.
3. **Navigation**: Implemented using React Navigation to manage transitions between screens and maintain navigation state.
4. **Redux State Management**: Centralized state management using Redux store, actions, and reducers to maintain application state and facilitate data flow between components.

The backend layer is built on Node.js with Express, providing RESTful API endpoints for the frontend to interact with. Key components include:

1. **Authentication Middleware**: Handles user authentication and authorization using JWT tokens.
2. **Route Handlers**: Process incoming requests, perform necessary operations, and return appropriate responses.
3. **Data Managers**: Encapsulate database operations and business logic for different data entities.
4. **External API Integration**: Connects with third-party services like NewsData API for content aggregation.

The application uses MongoDB, a NoSQL database, for data storage. Key aspects include:

1. **Collections**: Structured data storage for users, articles, events, comments, and other entities.
2. **Relationships**: Defined connections between different data entities, such as users and their created content.
3. **Indexes**: Optimized data retrieval for frequently accessed fields and query patterns.

Dynamic Architecture

Diagram:

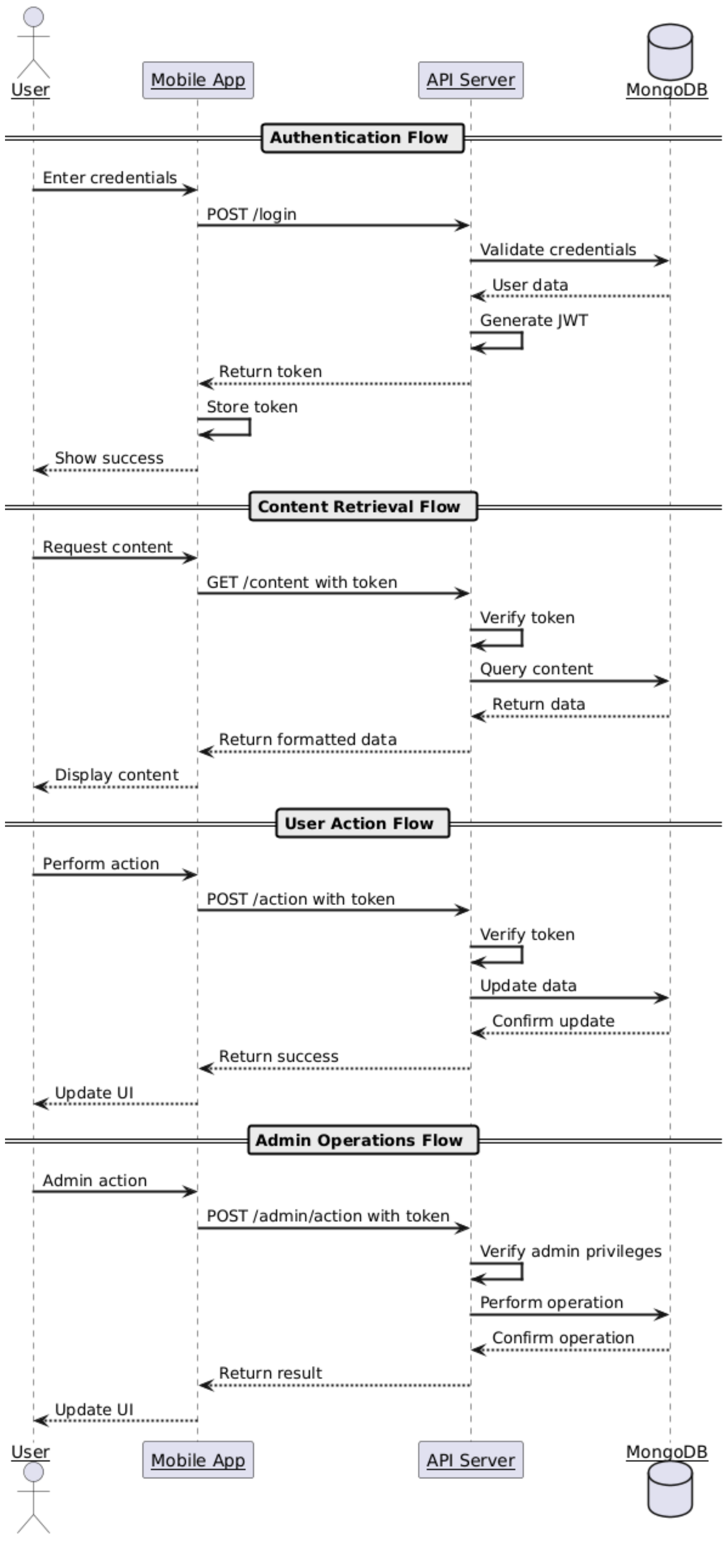


Figure 1.2: Dynamic Architecture

Description:
The dynamic system architecture provides a comprehensive view of the runtime behavior and interactions between system components. This architectural approach was selected to address several critical aspects of the system's operation.

In terms of user interaction flows, the architecture maps out complete sequences of user interactions throughout the system. It defines clear request-response patterns that occur between different components, ensuring predictable and reliable communication. The architecture also illustrates state changes and transitions that occur during user interactions, providing a clear understanding of how the system evolves in response to user actions. Additionally, it outlines error handling and recovery paths, ensuring the system can gracefully handle unexpected situations and maintain stability.

Key Interaction Patterns

1. **Authentication Flow**:
    - Client sends login credentials to the API
    - API validates credentials and generates JWT token
    - Token is returned to client and stored for future requests
    - Subsequent requests include token for authentication
2. **Content Retrieval Flow**:
    - Client requests content (articles, events, shop listings)
    - API authenticates request and retrieves data from database
    - Data is formatted and returned to client
    - Client updates UI with received data
3. **User Action Flow**:
    - User performs action (like article, save event, etc.)
    - Client sends action to API with authentication
    - API updates database and returns confirmation
    - Client updates UI to reflect the change
4. **Admin Operations Flow**:
    - Admin performs management action (user role change, content deletion)
    - Client sends request with admin authentication
    - API verifies admin privileges and performs operation
    - Database is updated and confirmation is returned
    - Client updates UI to reflect changes

State Management

The dynamic architecture incorporates Redux for state management, providing a predictable state container that maintains application data and UI state. This approach:

1. Centralizes application state for consistent access across components
2. Implements unidirectional data flow for predictable state updates
3. Facilitates debugging and testing through explicit state transitions
4. Enables optimized rendering through selective component updates

Error Handling and Recovery

The architecture includes robust error handling mechanisms:

1. API-level error detection and appropriate status code responses
2. Client-side error handling with user-friendly messages
3. Automatic token refresh for authentication failures
4. Graceful degradation for network connectivity issues
5. Retry mechanisms for transient failures

This comprehensive dynamic architecture ensures that the ATLSFW application delivers a responsive, reliable user experience while maintaining system integrity and data consistency.

Data Design

Introduction

This section outlines the data storage and exchange mechanisms within our system. The document details database structure, file usage, data exchange protocols, and security considerations, providing clarity for external developers. Our system primarily utilizes a **NoSQL MongoDB** database for flexible and scalable data management. The provided diagrams illustrate logical organization and relationships within our data.

Database Use

Database Selection and Structure:

Our system employs **MongoDB**, a NoSQL database, to efficiently store and manage data. This decision was made to support flexible schema design, scalability, and seamless integration with our mobile application. Below is a basic column family illustration of collections and their relationships to each other.

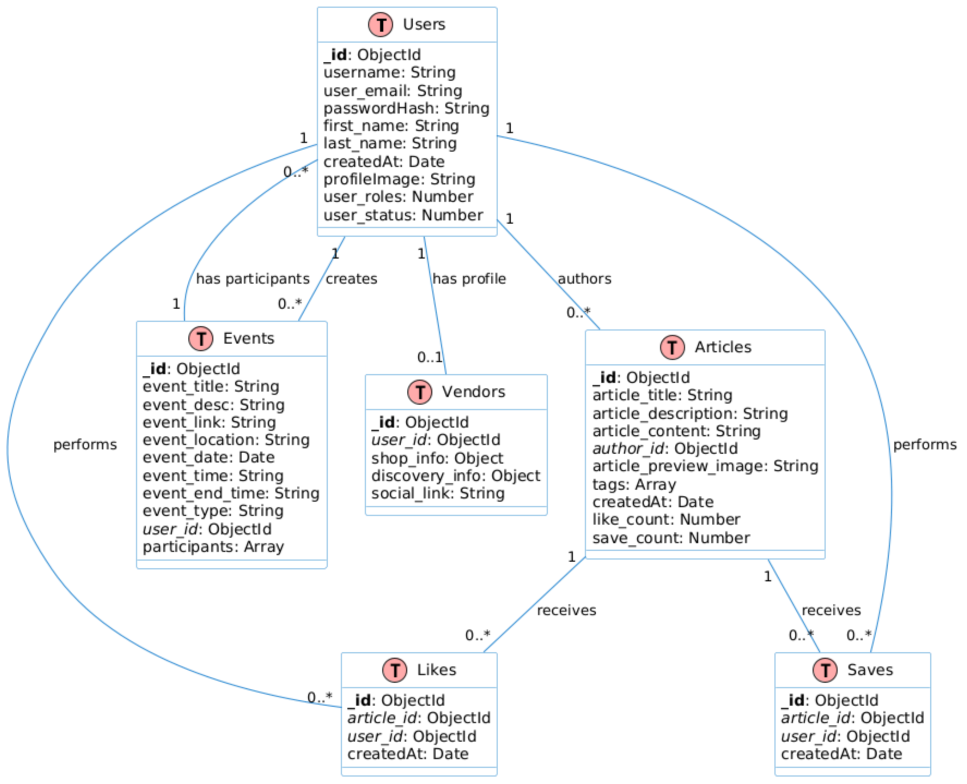


Figure 2.1: Graph Database Diagram

Collections and Data Structure:

1. **Users Collection**
   - Stores user account details, authentication data, and profile information.
   - References articles and comments made by users.

Field	**Type**	**Description**
_id	ObjectId	Unique identifier for each user
username	String	User's display name
email	String	User's email address
passwordHash	String	Hashed user password
createdAt	Date	Account creation date
profileImage	String	URL to user's profile image
user_roles	Number	Role Identifier (1: User, 2: Vendor, 3: Admin)
user_status	Number	Account status (1: Active, 0: Deactivated)

2. **Articles Collection**
   - Stores sustainable fashion-related articles.
   - References the user (author) who created the article.
   - Supports multiple comments.

Field	**Type**	**Description**
_id	ObjectId	Unique identifier for each article
article_title	String	Title of the article
article_description	String	Brief description of article
article_content	String	Main content of the article
authorId	ObjectId	Reference to the author's user _id
tags	Array	List of tags associated with article
createdAt	Date	Article publication date
like_count	Number	Number of likes received from users
save_count	Number	Number of times saved by users

3. **Events Collection**
   - Stores event details for sustainable fashion events.
   - References the user (admin or organizer) who created the event.

Field	**Type**	**Description**
event_title	String	Name of the event
event_desc	String	Description of the event
event_link	String	Link to outside website for the event
event_location	String	The location where an event is taking place
event_date	Date	The date of the event occurrence
event_time	String	Start time fo the event
event_end_time	String	End time of the event
user_id	ObjectId	Reference to uploader's user id
participants	Array	List of user IDs interested in the event

4. **Vendors Collection**
   - Extends user profiles with vendor-specific information
   - Includes shop details and discovery information for sustainable fashion brands

Field	**Type**	**Description**
_id	ObjectId	Unique identifier for each vendor
user_id	ObjectId	Reference to the user account
shop_info	Object	Contains shop details like name, description
discovery_info	Object	Contains brand information for discovery
social_link	String	Link to vendor's own website/social media

5. **Likes Collection**
    - Tracks user interactions on articles via likes

Field	Type	Description
_id	ObjectId	Unique identifier for each like
article_id	ObjectId	Reference to the liked article
user_id	ObjectId	Reference to the user who liked
createdAt	Date	When the like was created

6. **Saves Collection**
    - Tracks articles users have saved for later reading
    - Utilized to support personalized content libraries and engagement metrics

Field	Type	Description
_id	ObjectId	Unique identifier for each save
article_id	ObjectId	Reference to the saved article
user_id	ObjectId	Reference to the user who saved
createdAt	Date	When the save was created

Database Organization

Based on the application's database connection configuration, the data is organized into the following databases:

1. **posts_db**: Stores article content and related data
2. **saved_articles_db**: Manages user-saved articles
3. **users_db**: Contains user accounts and profile information
4. **news_db**: Stores news articles and related content
5. **events_db**: Manages event information and participation
6. **third_party_db**: Handles integration with external services

This organization provides logical separation of concerns while maintaining relationships between different data entities.

File Use

File documentation:
Our systems utilizes various file formats for configuration, data storage, and media content:

File Type	File Name	Format	Purpose
Configuration Files	package.json, mapping.json	JSON	Application settings and data mapping
Image Files	conn.png, db.png, uri.png	PNG	Data and URI structure diagrams
Lock Files	package-lock.json, yarn.lock	JSON, YAML	Dependency version control
Environment Files	environment_variables.mjs	MJS	Environment-specific configuration
Utility Scripts	*.mjs, *.js	JS, MJS	Utility functions and scripts

Data Exchange

Data exchange within the ATLSFW mobile application is managed through **RESTful API** over **HTTPS** to ensure secure communication. The API enables interactions between the frontend and backend for managing users, articles, comments and events. **JSON** is the standard data format used for API responses and internal configuration, providing structured and efficient data transfer.

API Structure

The API is organized into logical route groups:

1. **Authentication Routes** (/login, /signup): Handle user registration and authentication
2. **User Routes** (/user): Manage user profiles and preferences
3. **Article Routes** (/posts): Create, read, update, and delete articles
4. **Event Routes** (/events): Manage events and participation
5. **Vendor Routes** (/vendor): Handle vendor-specific operations
6. **Admin Routes** (/admin): Provide administrative functions

Data Format

JSON is the standard data format used for API responses and internal configuration, providing structured and efficient data transfer. Example response format:

```
json
{
  "status": "success",
  "data": {
    "articles": [
      {
        "_id": "60d21b4667d0d8992e610c85",
        "article_title": "Sustainable Cotton Alternatives",
        "article_description": "Exploring eco-friendly alternatives”
        "author_id": "60d21b4667d0d8992e610c80",
        "author_name": "Jane Smith",
        "tags": ["materials", "sustainability", "fashion"],
        "createdAt": "2025-03-15T14:30:00Z",
        "like_count": 42,
        "save_count": 18
      }
    ],
    "total": 1
  }
}
```

Security Considerations

Security is always a key concern in data storage and transmission. Measures include:

Data Encryption

- **Password Hashing**: User passwords are hashed using bcrypt before storage
- **Secure Communication**: All data exchanges occur over HTTPS to protect against interception
- **Sensitive Data Handling**: Personal Identifiable Information (PII) is stored securely with appropriate access controls

Authentication and Authorization

- **Token-based Authentication**: JSON Web Tokens (JWT) for secure session management
- **Role-based Access Control**: Different access levels for users, vendors, and administrators
- **Token Expiration**: Automatic token expiration with refresh mechanisms

Data Validation

- **Input Sanitization**: All user inputs are validated and sanitized before processing
- **Schema Validation**: Database operations validate data against defined schemas
- **Error Handling**: Comprehensive error handling to prevent data corruption

Compliance

- **GDPR Considerations**: The system follows best practices for GDPR and general security protocols
- **Data Minimization**: Only necessary data is collected and stored
- **User Consent**: Clear mechanisms for obtaining and managing user consent

Component Detailed Design

Introduction

The architecture of this system is designed to support a structured and efficient flow of information between different components, ensuring modularity and scalability. The system follows a well-defined component-based approach, maintaining conceptual integrity across both static and dynamic interactions. To provide a clear understanding of the system's structure and behavior, two key UML diagrams are presented: a static component diagram (Figure 3.1), which outlines the core components and their relationships, and a dynamic interaction diagram (Figure 3.2), which illustrates runtime interactions between these components. These diagrams reinforce the system's integrity by adhering to UML standards and demonstrating both the structural and behavioral aspects of the architecture.

Static

Figure 3.1 illustrates the static component diagram, providing an overview of the key components that form the foundation of the system. Each component is encapsulated with its defined functionality, attributes, and methods, ensuring a modular architecture that supports scalability and maintainability. The backend services include AuthMiddleware, responsible for verifying user authentication and authorization, and NewsDataAPI, which fetches and saves news articles in the DatabaseService. These components collectively manage secure data access and storage, ensuring a structured flow of information.

On the frontend, components such as AdminProfile, NavBar, and NewsFeedScreen facilitate user interactions. The AdminProfile component manages API requests for authentication and data retrieval, the NavBar component enables seamless navigation, and the NewsFeedScreen component displays fetched articles. Additionally, state management is handled by ReduxStore, which maintains essential system-wide states, including article data, authentication status, and user interactions. This ensures synchronization across UI components, providing a seamless user experience.

The relationships between these components are defined in a way that ensures clear dependencies and maintainability. The backend services interact with the frontend to fetch and store data, while the state management system maintains consistency in application behavior. Figure 3.1 adheres to UML component diagram standards, using appropriate notation to depict dependencies and functional relationships between system components.

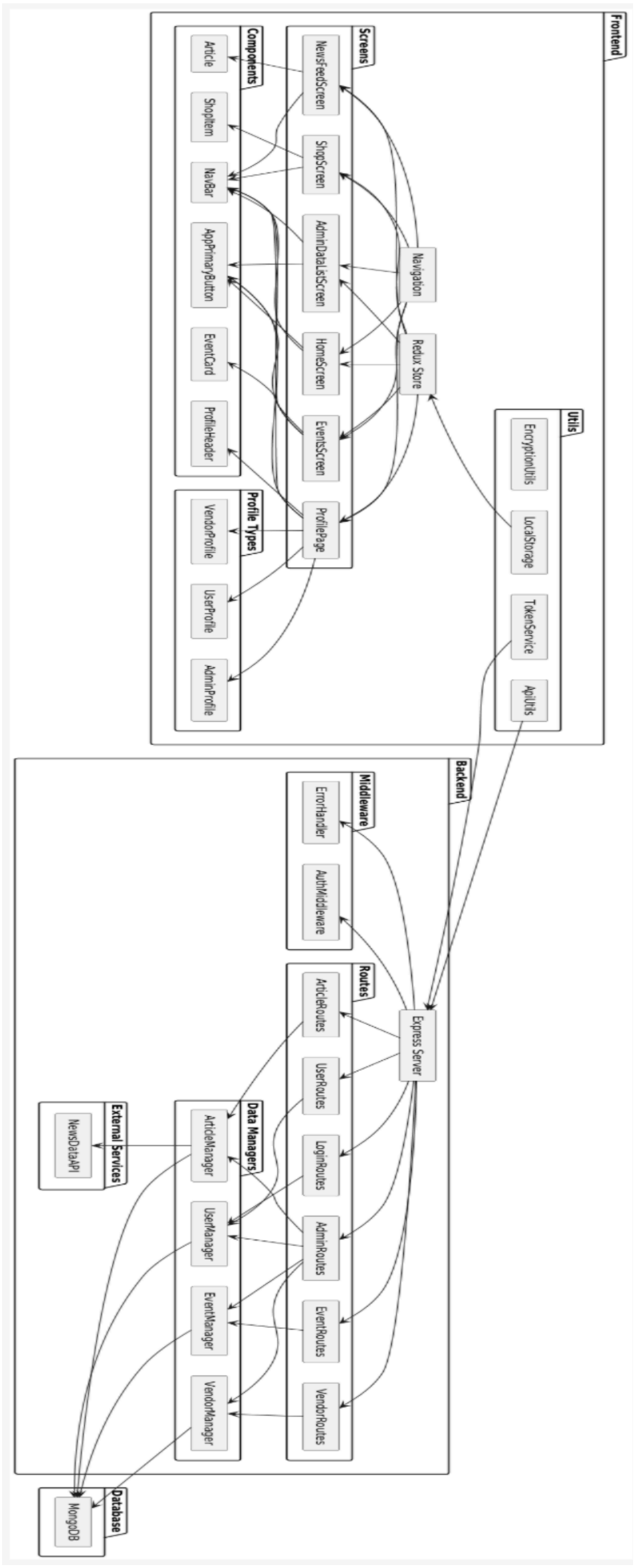


Figure 3.1: Static Component Design Diagram

Dynamic

Figure 3.2 illustrates the dynamic interaction of system components at runtime, capturing how different elements communicate to process user requests and system operations. The sequence begins with news feed configuration, where an admin user configures external data sources through NewsDataAPI. This component fetches articles, verifies admin authentication via AuthMiddleware, and stores the retrieved data in DatabaseService. This ensures that only authorized users can modify news sources and that all data modifications follow a secure protocol.

For general user interaction, the process starts when a user opens the news feed. The system verifies the user's authentication token using AuthMiddleware before proceeding with the request. Upon successful verification, the system retrieves the latest articles and updates the frontend display. Similarly, article interaction features, such as liking or saving articles, involve multiple components. When a user performs these actions, AuthMiddleware verifies their identity, updates interaction metrics, and confirms the changes in the system's database before reflecting them in the UI.

The final interaction sequence in Figure 3.2 showcases admin operations related to user management. When an admin modifies user roles, the system ensures authentication before applying the updates. The updated role is then stored in the database, and a confirmation response is sent back to the UI. This structured approach ensures that role modifications are securely handled and immediately reflected in the system. By depicting these runtime interactions, Figure 3.2 demonstrates how the system's static components behave dynamically during real-world operations, maintaining the conceptual integrity of the overall architecture.

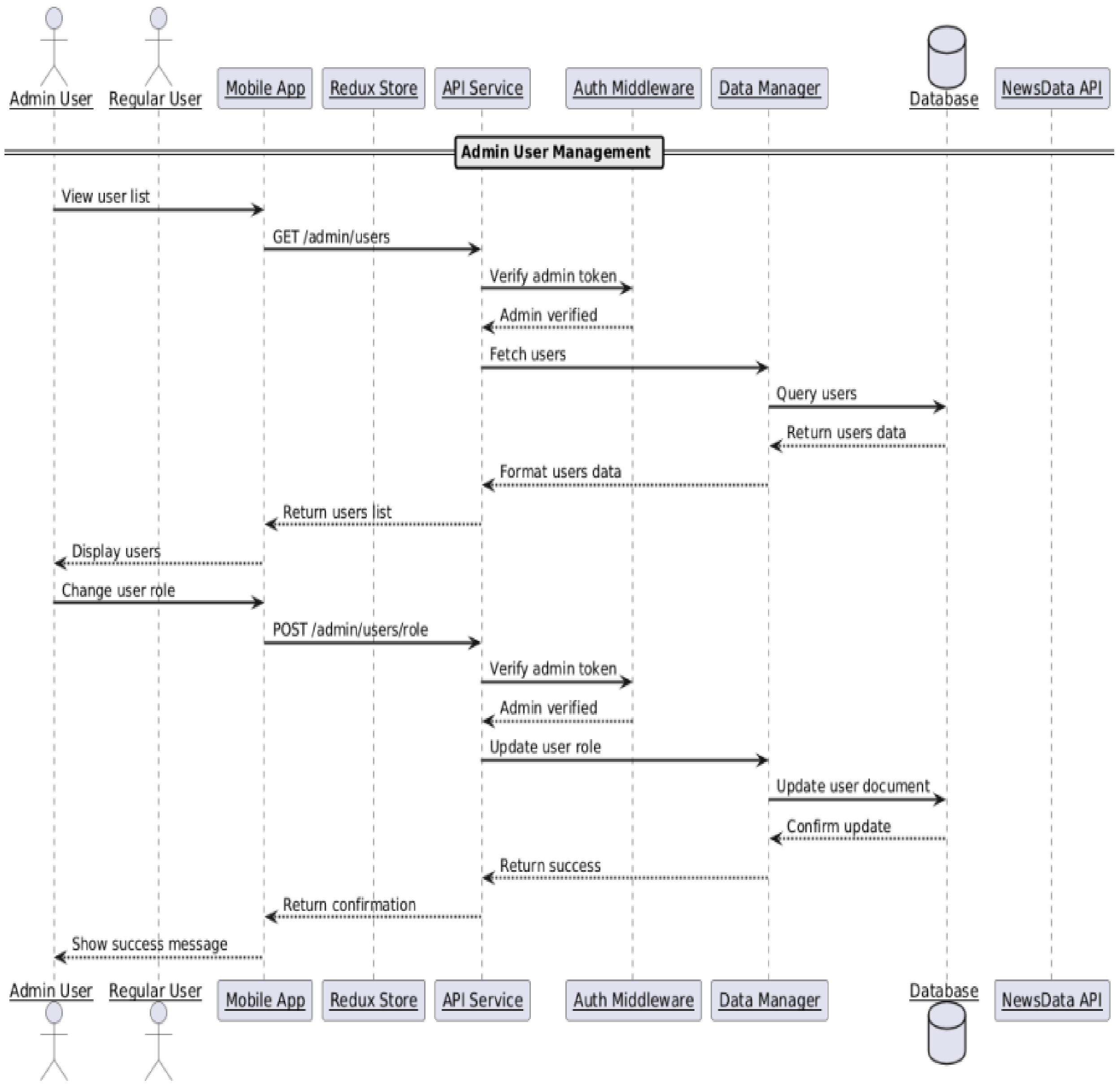


Figure 3.2: Dynamic Component Design Diagram

UI Design

Introduction

The user interface (UI) design of ATLSFW mobile application aims to provide an intuitive, seamless, and engaging user experience by showcasing sustainable fashion content, event listings, and profile features. This section of the Design Document examines the key screens users interact with, including the Home, Events, News Feed, Saved Articles, and Profile screens. Each screen is shown with a labeled figure, a textual description, and a heuristic analysis applying Nielsen's 10 Usability Heuristics.

Splash Screen

The splash screen is the first screen shown when the user opens the ATLSFW mobile app. It displays the brand's logo centered on a green background, reinforcing visual identity and signaling the app is loading.

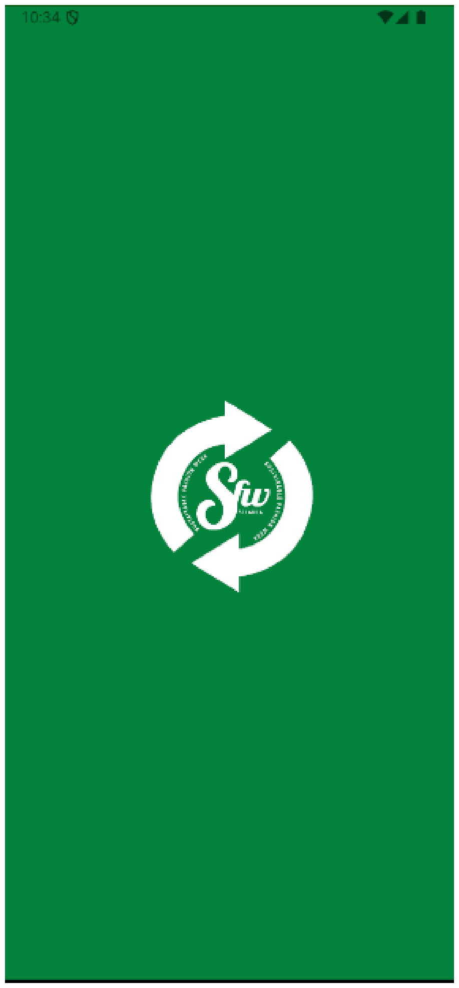


Figure 4.1: Splash Screen

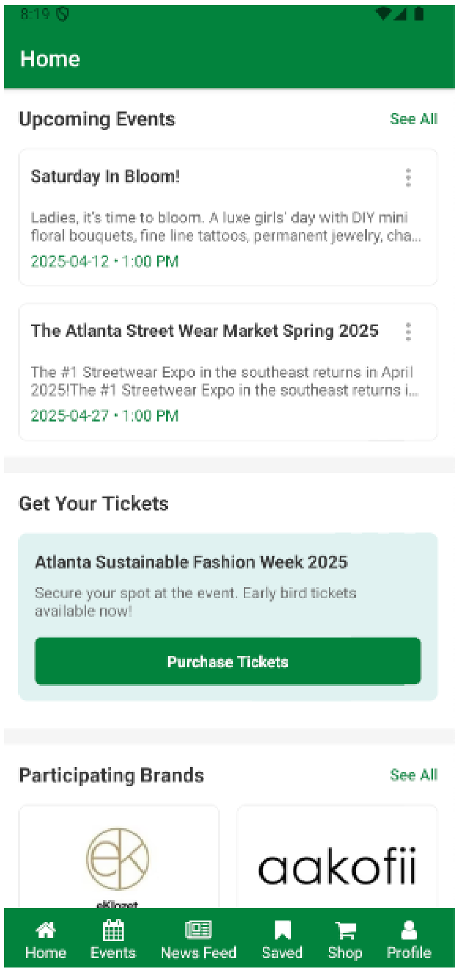
Heuristic Analysis:

- **Visibility of System Status:** The screen indicates that the app is loading or initializing, providing immediate feedback to the user.

- **Aesthetic and Minimalist Design:** The clean layout with a single logo on a green background avoids clutter while still conveying brand identity.
- **Match Between System and the Real World:** The use of official ATLSFW logo reinforces the legitimacy and branding of the organization.

Home Screen

The Home screen serves as the landing page. It highlights upcoming events, brand partners, and ticket purchasing options. It provides immediate access to featured content and serves as a visual overview for the platform's current highlights.


Home

Upcoming Events See All

Saturday In Bloom!

Ladies, it's time to bloom. A luxe girls' day with DIY mini floral bouquets, fine line tattoos, permanent jewelry, cha...

2025-04-12 • 1:00 PM

The Atlanta Street Wear Market Spring 2025

The #1 Streetwear Expo in the southeast returns in April 2025!The #1 Streetwear Expo in the southeast returns i...

2025-04-27 • 1:00 PM

Get Your Tickets

Atlanta Sustainable Fashion Week 2025

Secure your spot at the event. Early bird tickets available now!

Purchase Tickets

Participating Brands See All

Figure 4.2: Home Screen

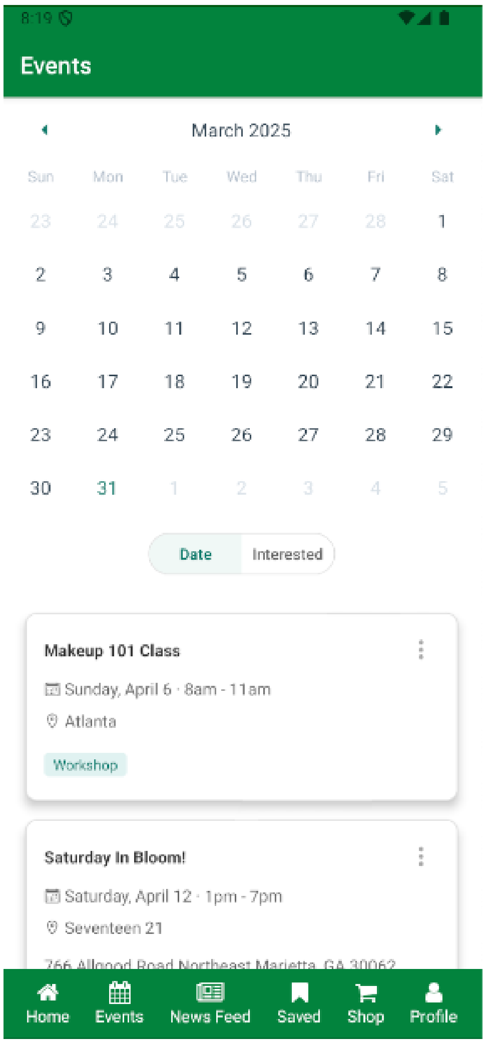
Heuristic Analysis:

- **Visibility of System Status:** The screen displays real-time updates like event titles, availability (e.g., "Now Live"), and partner highlights, keeping users informed about key activities.
- **Aesthetic and Minimalist Design:** The layout uses banners and whitespace effectively to focus user attention. Only essential information is shown, reducing clutter.

- **Consistency and Standards:** Navigation elements like icons and buttons align with platform conventions and are consistently used across the app.

Events Screen

The Events screen presents a calendar-based layout at the top with scrollable event cards below. Each event card includes the event name, time, location, and a “Save” or “Interested” toggle button.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
23	24	25	26	27	28	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Makeup 101 Class
Sunday, April 6 · 8am - 11am
Atlanta
Workshop

Saturday In Bloom!
Saturday, April 12 · 1pm - 7pm
Seventeen 21
766 Allgood Road Northeast Marietta, GA 30062

Figure 4.3: Events Screen

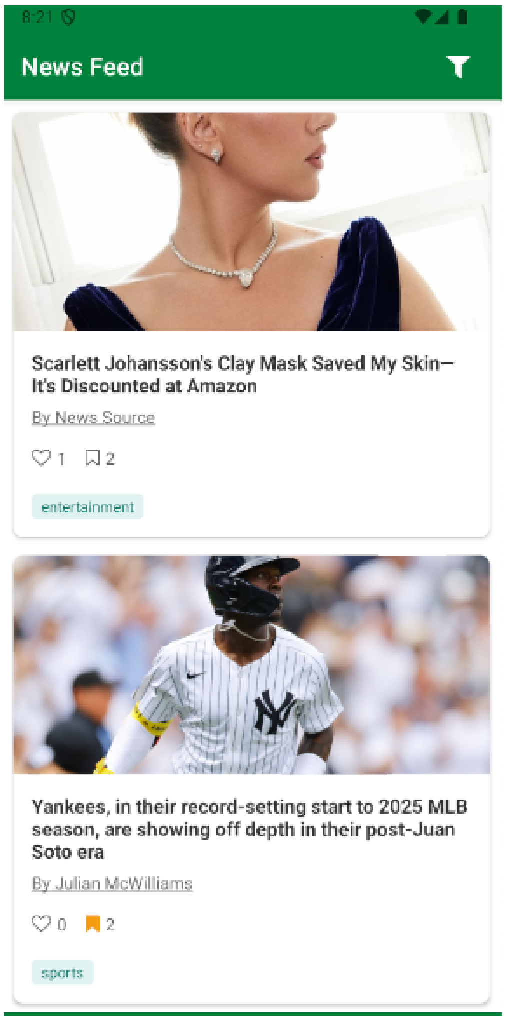
Heuristic Analysis:

- **Match Between System and the Real World:** The calendar format mimics real-world scheduling tools, making it intuitive for planning.
- **Recognition Rather Than Recall:** Events are visible directly below the calendar with details like date, time, and location — allowing users to quickly identify and select events without needing to remember them.

- **Flexibility and Efficiency of Use:** The “Date” and “Interested” filters help users toggle between all upcoming events and those they’ve marked, supporting both exploration and personalized navigation.

<u>News Feed Screen</u>

The News Feed screen provides a vertically scrollable feed of articles, each presented in a card format. Each card includes a large header image, article title, author/source, and interaction icons for likes, saves, and comments. Articles are also labeled by category (e.g., “entertainment,” “sports”), and a filter icon appears in the top right corner for sorting or filtering content.


8:21

News Feed

Scarlett Johansson's Clay Mask Saved My Skin—It's Discounted at Amazon

By News Source

♡ 1 2

entertainment

Yankees, in their record-setting start to 2025 MLB season, are showing off depth in their post-Juan Soto era

By Julian McWilliams

♡ 0 2

sports

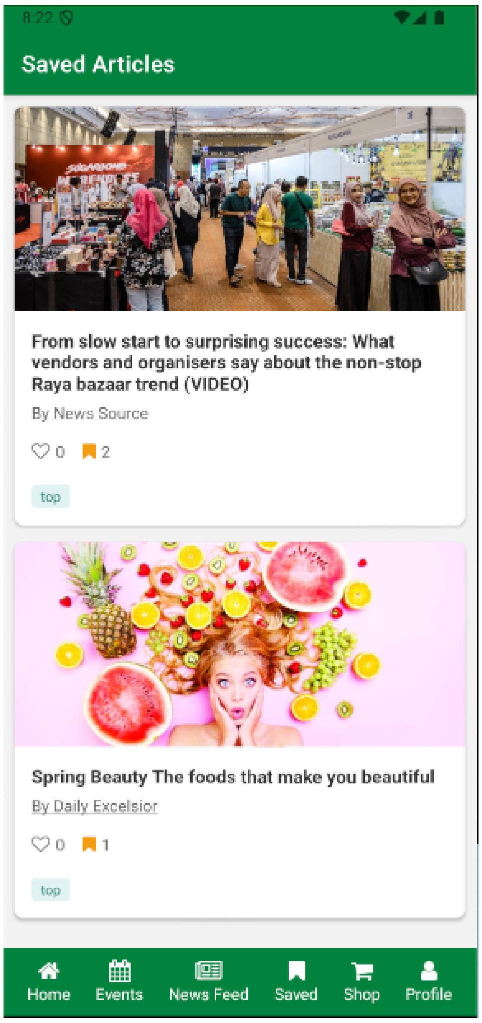
Figure 4.4 - News Feed Screen

Heuristic Analysis:

- **Recognition Rather Than Recall:** Key information (title, author, category, image) is visually presented within each card. This reduces cognitive load by allowing users to recognize relevant content at a glance.
- **User Control and Freedom:** Users can engage with articles freely — scrolling, clicking on items, or using the top-right filter icon to refine their view.
- **Consistency and Standards:** Interaction icons (heart for like, comment bubble, bookmark) follow widely recognized design conventions, making their purpose immediately understandable.

Saved Articles Screen

This screen displays articles the user has saved for later reading. Cards display header image, titles, and publisher names, with action icons for liking or unsaving.


Saved Articles

From slow start to surprising success: What vendors and organisers say about the non-stop Raya bazaar trend (VIDEO)

By News Source

0 2

top

Spring Beauty The foods that make you beautiful

By Daily Excelsior

0 1

top

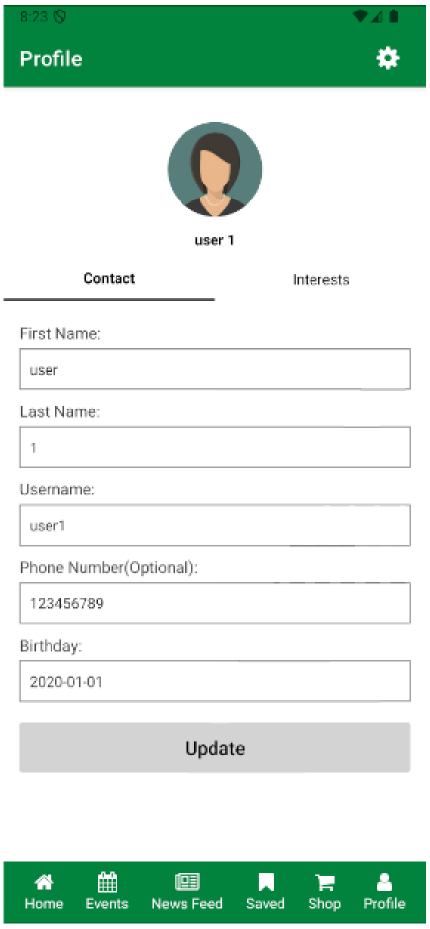
Figure 4.5 - Saved Articles Screen

Heuristic Analysis:

- **Consistency and Standards:** Reuses the card layout from the News Feed, maintaining uniformity in presentation and interaction design.
- **Flexibility and Efficiency of Use:** Enables quick access to previously saved content, improving efficiency for returning users.
- **Error Prevention:** Clear labels and icons help prevent unintended interactions such as accidental unsaving.

Profile Screen

The Profile screen enables users to manage and update their personal information. At the top, the screen displays a user avatar and username. It includes editable fields for first name, last name, username, phone number (optional), and birthday, all laid out in a clean vertical form. A large "Update" button at the bottom allows users to save their changes. A gear icon in the top-right corner suggests additional profile settings or preferences.


8:23
Profile
user 1

First Name: user
Last Name: 1
Username: user1
Phone Number(Optional): 123456789
Birthday: 2020-01-01

Figure 4.6 - Profile Screen

Heuristic Analysis:

- **Visibility of System Status:** Each field is pre-filled or shows placeholder text, clearly indicating which values can be edited. The "Update" button provides an obvious call to action.
- **Help and Documentation:** Field labels and placeholder texts guide the user through expected inputs, minimizing guesswork and confusion.
- **Aesthetic and Minimalist Design:** The screen uses a focused layout with adequate spacing between elements, avoiding clutter while prioritizing usability and readability.

Shop Screen

The Shop screen displays a list of sustainable fashion vendors and their products. Each shop card includes the brand name, an image or logo, and options to visit their website or social media. Admin users have additional options to manage shop listings.

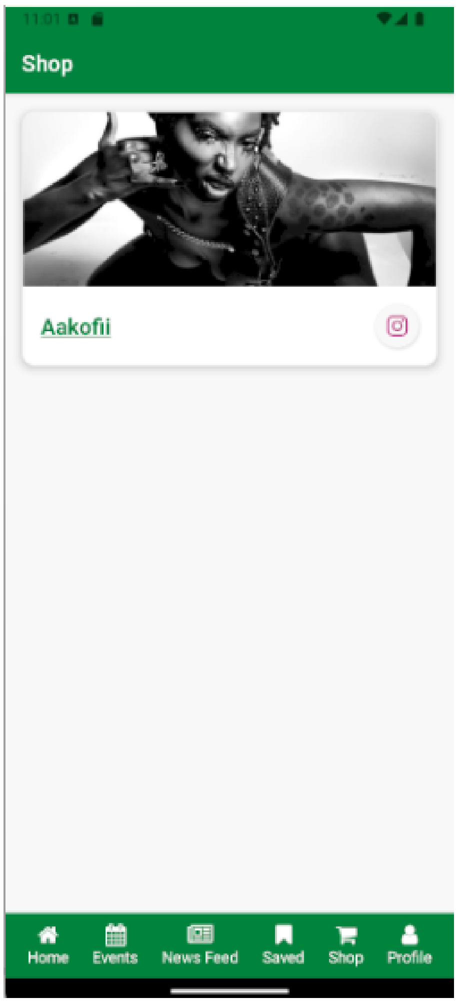


Figure 4.7 - Shop Screen

Heuristic Analysis

- Visibility of System Status: Each shop card clearly indicates whether it has a website or social media links available through the action buttons.
- Match Between System and the Real World: The shop layout resembles a directory or marketplace, making it familiar to users accustomed to online shopping.
- User Control and Freedom: Users can easily navigate between shops and access external links to learn more about each vendor.
- Consistency and Standards: The shop cards follow a consistent design pattern, making it easy for users to scan and compare different vendors.

Admin Data List Screen

The Admin Data List screen provides administrative functions for managing users, vendors, articles, and analytics. It displays lists of data with options to filter, sort, and perform administrative actions such as changing user roles, activating/deactivating accounts, and deleting content.

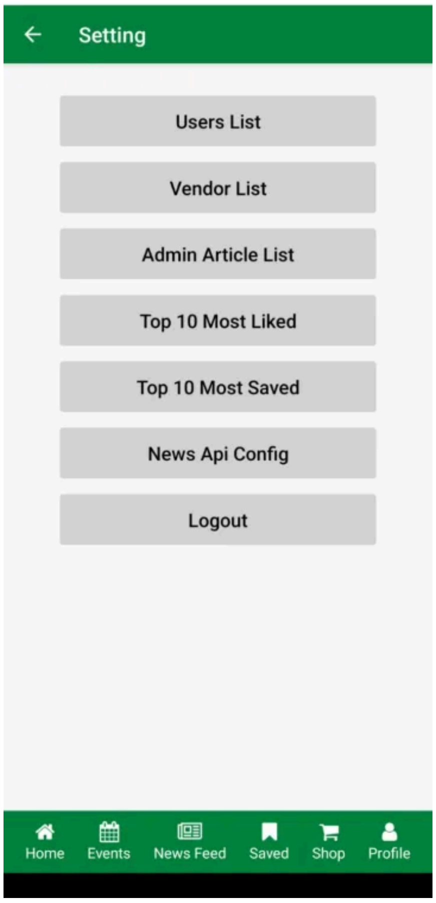


Figure 4.8 - Admin List Screen

Heuristic Analysis

- **Visibility of System Status**: User cards clearly display current role and status, making it easy to identify which users are active and what roles they have.
- **User Control and Freedom**: The options menu provides administrative actions with confirmation dialogs to prevent accidental changes.
- **Recognition Rather Than Recall**: All relevant user information is displayed on the card, eliminating the need to remember details.
- **Flexibility and Efficiency of Use**: The list view allows administrators to quickly scan and manage multiple users, with options to filter and sort for efficiency.

Design Considerations

- **Widget Selection:** Standard UI components (e.g., buttons, date pickers, navigation tabs) were selected to maintain cross-platform familiarity and accessibility.
- **Layout:** Screens are structured with visual hierarchy — primary actions (like Save or Like) appear prominently, while secondary information (like author names) is de-emphasized.
- **Visual Feedback:** Real-time UI feedback (e.g., button toggles, loading animations) confirms user interactions and reinforces trust.
- **Navigation:** Consistent bottom tab navigation ensures a familiar flow across screens.
- **Responsiveness:** Layouts are designed to adapt to various screen sizes while maintaining readability and touch target size.

Appendix

NewsData API

Resource URL:
https://newsdata.io/api/1/latest?apikey=YOUR_API_KEY

Resource Information:
Response Format: JSON
Requires Authentication: Yes
Rate Limit: 2,000 articles/day

Parameters:
apikey: our API key
id: search by article ID
q: search articles by specific keywords
qInTitle: search articles for keywords only in the title
country: search news articles from specific countries

Example Request:
https://newsdata.io/api/1/latest?apikey=YOUR_API_KEY&country=au,us

Example Response:

```
json
{
  "status": "success",
  "totalResults": 31698,
  "results": [
    {
      "article_id": "78b6cfb16118a9e8b1a097de9c0f1036",
      "title": "Gerichte heben nur noch relativ wenige auf",
      "link": "https://www.wn.de/welt/politik",
      "keywords": null,
      "creator": null,
      "video_url": null,
      "description": "Mehr als jeder dritte Schutzsuchende
      "content": "Die staatlich geförderte kostenlose",
      "pubDate": "2025-03-30 06:32:58",
      "pubDateTZ": "UTC",
      "image_url": "https://asc-images.forward",
      "source_id": "wn",
      "source_priority": 35525,
      "source_name": "Westfälische Nachrichten",
      "source_url": "https://www.wn.de/",
      "source_icon": "https://i.bytvi.com/domain_icons/wn.png",
      "language": "german",
      "country": ["germany"],
      "category": ["top"],
      "ai_tag": ["awards and recognitions"],
      "ai_region": null,
      "ai_org": null,
      "sentiment": "neutral",
      "sentiment_stats": {
        "positive": 0.12,
        "neutral": 99.74,
        "negative": 0.14
      },
      "duplicate": true
    }
  ]
}
```

API Endpoints

The ATLSFW application uses a RESTful API architecture with the following key endpoints:

Authentication Endpoints

Login

- **URL**: /login
- **Method**: POST
- **Description**: Authenticates a user and returns a JWT token

Request Body:

```
json
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

Response:

```
json
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "_id": "60d21b4667d0d8992e610c80",
    "username": "johndoe",
    "user_roles": 1,
    "user_status": 1
  }
}
```

Signup

- **URL**: /signup
- **Method**: POST
- **Description**: Registers a new user

Request Body:

```
json
{
  "username": "johndoe",
  "email": "user@example.com",
  "password": "securepassword",
  "first_name": "John",
  "last_name": "Doe"
}
```

Response:

```
json
{
  "success": true,
  "message": "User registered successfully",
  "user_id": "60d21b4667d0d8992e610c80"
}
```

Article Endpoints

Get Articles

- **URL**: /posts
- **Method**: GET
- **Description**: Retrieves a list of articles
- **Query Parameters**:
    - limit: Number of articles to return (default: 10)
    - page: Page number for pagination (default: 1)
    - tags: Filter by tags (comma-separated)

Response:

```
json
{
  "success": true,
  "articles": [
    {
      "_id": "60d21b4667d0d8992e610c85",
      "article_title": "Sustainable Cotton Alternatives",
      "article_description": "Exploring eco-friendly alternatives",
      "author_id": "60d21b4667d0d8992e610c80",
      "author_name": "Jane Smith",
      "tags": ["materials", "sustainability", "fashion"],
      "createdAt": "2025-03-15T14:30:00Z",
      "like_count": 42,
      "save_count": 18
    }
  ],
  "total": 1,
  "page": 1,
  "limit": 10
}
```

Create Article

- **URL**: /posts
- **Method**: POST
- **Description**: Creates a new article
- **Authentication**: Required

Request Body:

```
json
{
  "article_title": "Sustainable Cotton Alternatives",
  "article_description": "Exploring eco-friendly alternatives",
  "article_content": "Detailed content about sustainable cotton",
  "tags": ["materials", "sustainability", "fashion"],
  "article_preview_image": "https://example.com/images/.jpg"
}
```

Response:

```
json
{
  "success": true,
  "message": "Article created successfully",
  "article_id": "60d21b4667d0d8992e610c85"
}
```

Event Endpoints

Get Events

- **URL**: /events
- **Method**: GET
- **Description**: Retrieves a list of events
- **Query Parameters**:
    - event_type: Filter by event type (e.g., "regular", "workshop")
    - date: Filter by specific date (YYYY-MM-DD)
    - month: Filter by month (YYYY-MM)

Response:

```
json
{
  "success": true,
  "events": [
    {
      "_id": "60d21b4667d0d8992e610c90",
      "event_title": "Sustainable Fashion Show",
      "event_desc": "Showcasing the latest in sustainable
fashion",
      "event_location": "Atlanta Convention Center",
      "event_date": "2025-04-20",
      "event_time": "19:00",
      "event_end_time": "22:00",
      "event_type": "regular",
      "user_id": "60d21b4667d0d8992e610c80",
      "participants": ["60d21b4667d0d8992e610c81"]
    }
  ]
}
```

Create Event

- **URL**: /events
- **Method**: POST
- **Description**: Creates a new event
- **Authentication**: Required (Admin only)

Request Body:

```
json
{
  "event_title": "Sustainable Fashion Show",
  "event_desc": "Showcasing the latest in sustainable fashion",
  "event_location": "Atlanta Convention Center",
  "event_date": "2025-04-20",
  "event_time": "19:00",
  "event_end_time": "22:00",
  "event_type": "regular",
  "event_link":
"https://example.com/events/sustainable-fashion-show"
}
```

Response:

```
json
{
  "success": true,
  "message": "Event created successfully",
  "event_id": "60d21b4667d0d8992e610c90"
}
```

Add Participant

- **URL**: /events/participant
- **Method**: POST
- **Description**: Adds a user as interested in an event
- **Authentication**: Required

Request Body:

```
json
{
  "event_id": "60d21b4667d0d8992e610c90"
}
```

Response:

```
json
{
  "success": true,
  "message": "Added to event participants"
}
```

Shop/Vendor Endpoints

Get Vendors

- **URL**: /vendors
- **Method**: GET
- **Description**: Retrieves a list of vendors/shops

Response:

```
json
{
  "success": true,
  "vendors": [
    {
      "_id": "60d21b4667d0d8992e610c95",
      "user_id": "60d21b4667d0d8992e610c85",
      "shop_info": {
        "brand_name": "EcoFashion",
        "shop_now_link": "https://ecofashion.example.com",
        "url": "https://example.com/images/ecofashion-banner.jpg"
      },
      "discovery_info": {
        "title": "Sustainable clothing made from recycled
materials",
        "intro": "EcoFashion creates stylish, sustainable cloth"
      },
      "social_link": "https://instagram.com/ecofashion"
    }
  ]
}
```

Create Vendor

- **URL**: /vendors
- **Method**: POST
- **Description**: Creates a new vendor profile for a user
- **Authentication**: Required (Admin only)

Request Body:

```
json
{
  "user_id": "60d21b4667d0d8992e610c85",
  "shop_info": {
    "brand_name": "EcoFashion",
    "shop_now_link": "https://ecofashion.example.com",
    "url": "https://example.com/images/ecofashion-banner.jpg"
  },
  "discovery_info": {
    "title": "Sustainable clothing made from recycled materials",
    "intro": "EcoFashion creates stylish, sustainable cloth"
  },
  "social_link": "https://instagram.com/ecofashion"
}
```

Response:

```
json
{
  "success": true,
  "message": "Vendor profile created successfully",
  "vendor_id": "60d21b4667d0d8992e610c95"
}
```

Update Vendor

- **URL**: /vendors/
- **Method**: PUT
- **Description**: Updates an existing vendor profile
- **Authentication**: Required (Vendor or Admin)

Request Body:

```
json
{
  "shop_info": {
    "brand_name": "EcoFashion Updated",
    "shop_now_link": "https://ecofashion-updated.example.com",
    "url": "https://example.com/images/ecofashion-new-banner.jpg"
  }
}
```

Response:

```
json
{
  "success": true,
  "message": "Vendor profile updated successfully"
}
```

Delete Vendor

- **URL**: /vendors/
- **Method**: DELETE
- **Description**: Deletes a vendor profile
- **Authentication**: Required (Admin only)

Response:

```
json
{
  "success": true,
  "message": "Vendor profile deleted successfully"
}
```

User Management Endpoints

Change User Role

- **URL**: /admin/users/role
- **Method**: POST
- **Description**: Changes a user's role (e.g., from regular user to vendor)
- **Authentication**: Required (Admin only)

Request Body:

```
json
{
  "user_id": "60d21b4667d0d8992e610c85",
  "role": 2
}
```

Response:

```
json
{
  "success": true,
  "message": "User role updated successfully"
}
```

Deactivate User

- **URL**: /admin/users/status
- **Method**: POST
- **Description**: Activates or deactivates a user account
- **Authentication**: Required (Admin only)

Request Body:

```
json
{
  "user_id": "60d21b4667d0d8992e610c85",
  "status": 0
}
```

Response:

```
json
{
  "success": true,
  "message": "User status updated successfully"
}
```

Error Handling

The ATLSFW application implements a comprehensive error handling strategy to ensure a robust and user-friendly experience. This section outlines the approach to error handling across different components of the system.

API Error Handling

All API endpoints follow a consistent error handling pattern:

1. **HTTP Status Codes**: Appropriate HTTP status codes are used to indicate the nature of errors:

- 400: Bad Request (invalid input)
- 401: Unauthorized (missing or invalid authentication)
- 403: Forbidden (insufficient permissions)
- 404: Not Found (resource doesn't exist)
- 500: Internal Server Error (unexpected server issues)

2. **Error Response Format**: All error responses follow a consistent JSON format:

```
json
{
  "success": false,
  "error": {
    "code": "ERROR_CODE",
    "message": "Human-readable error message",
    "details": {} // Optional additional information
  }
}
```

3. **Validation Errors**: For validation errors, the details field includes specific validation failures:

```
json
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": {
      "email": "Must be a valid email address",
      "password": "Must be at least 8 characters long"
    }
  }
}
```

Client-Side Error Handling

The mobile application implements several strategies for handling errors:

1. **Network Error Detection**: The application detects network connectivity issues and provides appropriate feedback to users.
2. **Error Boundaries**: React error boundaries catch JavaScript errors in UI components, preventing the entire application from crashing.
3. **Toast Notifications**: Non-critical errors are displayed as toast notifications that provide information without disrupting the user experience.
4. **Modal Dialogs**: Critical errors that prevent further action are displayed in modal dialogs with clear instructions for resolution.
5. **Offline Support**: For network-related errors, the application falls back to cached data when possible and queues write operations for later execution.

Error Logging

Error Logging

The application implements comprehensive error logging to facilitate debugging and issue resolution:

1. **Server-Side Logging**: All API errors are logged with relevant context information, including:
   - o Timestamp
   - o Error type and message
   - o Request details (endpoint, method, parameters)
   - o User information (if authenticated)
   - o Stack trace

2. **Client-Side Logging**: The mobile application logs errors to a remote service, capturing:
   - Device information
   - Application version
   - User actions leading to the error
   - Stack trace
   - Network request details
3. **Error Aggregation**: Logs are aggregated and analyzed to identify patterns and prioritize fixes.

Development Environment Setup

This section provides instructions for setting up the development environment for the ATLSFW mobile application.

Prerequisites

- Node.js (v14.x or later)
- npm (v6.x or later) or Yarn (v1.22.x or later)
- MongoDB (v4.4.x or later)
- Expo CLI (v4.x or later)
- Android Studio (for Android development)
- Xcode (for iOS development, macOS only)

Backend Setup

1. Clone the repository:

```
bash
git clone
https://github.com/KTang603/JID-4347-ATLSFW-App-Enhancement.git
cd JID-4347-ATLSFW-App-Enhancement
```

2. Install server dependencies:

```
bash
cd server
npm install
```

3. Configure environment variables:

```
bash
cp .env.example .env
# Edit .env with your MongoDB connection string and JWT secret

#Eaxmple

# Database Configuration
```

```
MONGODB_URI=mongodb+srv://username:password@cluster0.example.mongo
db.net/?retryWrites=true&w=majority

# Authentication
JWT_SECRET=your_secure_random_string_here
JWT_EXPIRATION=24h

# Application Settings
PORT=5050
NODE_ENV=development
```

4. Start the server:

```
bash
npm run dev or npm start
```

Frontend Setup

1. Install client dependencies:

```
bash
cd client
npm install
```

2. Configure environment variables:

```
bash
cp client/environment_variables.example.mjs
client/environment_variables.mjs
# Edit environment_variables.mjs with your API endpoint

#Example

#Local IP address for development
const MY_IP_ADDRESS = "your.local.ip.address";

export default MY_IP_ADDRESS;
```

3. Start the Expo development server:

```
bash
npm start
```

4. Run on a device or emulator:
   - Press 'a' to run on Android emulator
   - Press 'i' to run on iOS simulator (macOS only)
   - Scan the QR code with the Expo Go app on your physical device

Testing

1. Run backend tests:

```
bash
cd server
npm test
```

2. Run frontend tests:

```
bash
cd client
npm test
```

Contact information

Vivek Bumb
vbumb3@gatech.edu

- Backend Development
- API Integration
- Deployment and CI/CD
- Testing &QA

Kevin Tang
ktang80@gatech.edu

- Backend Development
- Testing &QA
- Deployment and CI/CD

Xilu Zeng
xzeng97@gatech.edu

- Backend Development
- UI/UX Design

Ruixuan Gao
rgao68@gatech.edu

- Frontend Engineering
- UI/UX Design
- Presentation and Client Communication

Siddhant Agarwal
sagarwal437@gatech.edu

- Frontend Engineering
- Testing & QA
- API Integration