SW Engineering CSC648-848 Spring 2025

Project/application title and name: Gator Market

Section 04 Team 1 Milestone 4

Name	Role	Email
Dev Modi	Team Lead, Backend, Frontend, database	dmodi@sfsu.edu
Yash Pachori	Backend Lead with Database	ypachori@sfsu.edu
Kyle Yuen	Frontend Lead	kyuen4@sfsu.edu
Hsueh-Ta Lu	Scrum Master with Frontend	hlu@sfsu.edu
Daniel	Tech Lead and Github	domstead@sfsu.edu

Revision History

Date	**Notes**
30 March 2025	Initial publication of M4 document

Topic Index

1) Product Summary

Product Name: Gator Market

Product Description:

Gator Market is a secure, student-exclusive online marketplace built specifically for San Francisco State University (SFSU). Its primary purpose is to make buying and selling items—like textbooks, dorm supplies, and electronics—easy, safe, and trustworthy within the SFSU community. By requiring users to sign up with verified @sfsu.edu emails and including built-in chat messaging, the platform eliminates risks found on platforms like Craigslist or Facebook Marketplace. Unique to Gator Market are AI-powered pricing suggestions and group-buying tools, making it smarter and more budget-friendly than traditional marketplaces. The platform is student-built and designed to solve real problems that students face daily, turning campus transactions into a safer, smarter, and community-driven experience.

Final List of Committed (P1) Functionalities:

These are the features our team commits to deliver and test fully. Each line is a complete, plain-English function that will be available and functional at launch.

For Unregistered Users:

- Can view all item listings.
- Can use search and filter by category, keyword, price, and condition.

For Registered Users:

- Can register using a verified SFSU email address.
- Can log in, log out, and manage account settings.
- Can create, edit, and delete their product listings.

- Can upload one or more images when posting products.
- Can send and receive secure messages to/from other users within the app.
- Can view messages and respond to them.
- Can receive email or in-app alerts when new listings match their interests.
- Can report suspicious or inappropriate listings or users.
- Can leave ratings and written reviews for other users after a transaction.

Deployment URL:

http://csc648g1.me

2) Usability test plan for selected function

1. Test Objectives

Goal: Ensure users can easily upload a product, fill in necessary details (like product name, price, description), upload an image, and use the safety feature (allowing photo/video recording during meetups).

- Test if users can complete the upload process without issues.
- Check if the safety option (photo/video recording for safety) is clear and understandable.

---

2. Test Background and Setup

System Setup:

- Testing will happen on desktop browsers, using our live production server.

Testers:

- Regular users with no technical background.
- They should be able to fill in a form and upload a product.

Testing Environment:

- Tests will be done from home or office with a stable internet connection and desktop or laptop devices.

3. Usability Task Description

Tasks:

1. Go to the product upload page.
2. Fill in these fields:
   - Product name
   - Description
   - Price
   - Category
   - Condition
3. Upload a product image (max 5MB).
4. Decide if you'd like to allow photo/video recording for safety during meetups (optional).
5. Submit the form and check for a success message.

4. Effectiveness and Efficiency Evaluation

Effectiveness:

- Can users complete the product upload without help?
- Do they understand the safety option (photo/video recording)?
- Is the image upload feature working smoothly?

Efficiency:

- How long does it take to complete the upload task?
- Are there any issues or confusion while filling out the form?
- Do users easily understand the safety feature?

5. User Satisfaction

After the task, ask users to rate the following on a Likert scale (1 = Very Unsatisfied, 5 = Very Satisfied):

1. How easy was the product upload process?
2. How satisfied are you with the form design and layout?
3. How did you feel about the image upload feature (e.g., preview)?
4. How clear were the error messages (if any)?
5. How comfortable are you with the option to allow photo/video recording during meetups for safety?

---

6. Use of GenAI

- Tool Used: We used ChatGPT to help generate and refine this test plan.
- How It Was Used: ChatGPT made sure the test objectives, tasks, and questions were clear, concise, and comprehensive.
- Feedback: It helped improve the flow of the tasks and clarified the safety feature's role in the test.

---

7. Conclusion

This plan ensures that the product upload feature is user-friendly, including the safety option for photo/video recording. We’ll check:

- Whether users can upload a product with no issues.
- If they understand and use the safety feature.
- How satisfied they are with the process overall.

3) QA Test Plan and QA Testing Report: Gator Market - Homepage Search & Filter

1. Test Objectives

To verify that the search and category filter system on the homepage:

- Correctly fetches and displays products based on user input
- Maintains a history of recent searches
- Clears and updates product results dynamically
- Functions consistently across different browsers

2. Hardware and Software Setup

Hardware:

- PC/Laptop (Windows 11/macOS)

Software:

- Web Browser: Google Chrome

URL:

- VM Ware
- VS Code

3. Features to be tested:

- Product search bar functionality
- Category dropdown filter
- Search history display and clearing
- Fetching and rendering filtered product results

4. Q&A Test Plan

Test #	Test Title	Description	Input	Expected Output	Pass/Fail
1	Initial Product Display	The product should be listed	Page loads, no input needed	Products should be there	Pass
2	Keyword Search	Test product search using the keyword	Enter "laptop" or "computer"	Takes you to your input	Pass
3	Category Filter	Filter products by category	Select "phones"	Should show all phones	Pass
4	Search History	Verifies local storage and saves the search history	Search phone, then reload the page	phone	Pass
5	Bookmark	Tests to see if we can bookmark an item	Click the bookmark emblem	The product appears in the bookmarks	Pass
6	Remove Bookmark	Tests to see if we can remove a bookmarked item	Click the bookmark emblem again so its not highlight	The bookmark item should no longer be in bookmarks	Pass

5. Multi-Browser Testing Results

- Google Chrome (pass)
- Safari (pass)

6. GenAI use

GenAI tools, including ChatGPT, were actively used throughout the QA testing process to:

- Generate detailed and structured test case templates
- Refine test descriptions and expected outcomes for clarity and completeness
- Suggest additional edge cases for broader test coverage
- Assist in drafting automated test scripts for search and filter functionalities
- Provide real-time feedback on UI/UX elements based on usability principles

The use of GenAI significantly accelerated the documentation process and helped ensure consistency and thoroughness in testing.

4) Peer Code Review

1. **Peer Review #1: Yash (Backend Lead) Reviewed Dev’s Frontend Code (AdminDashboard.jsx)**

Code Reviewed:

- File: AdminDashboard.jsx
- Feature: Admin interface for managing product approvals, users, and viewing reports

Review Request:

Dev shared the GitHub pull request feature/admin-dashboard-ui and requested a formal review from Yash for usability, API integration, and error handling feedback.

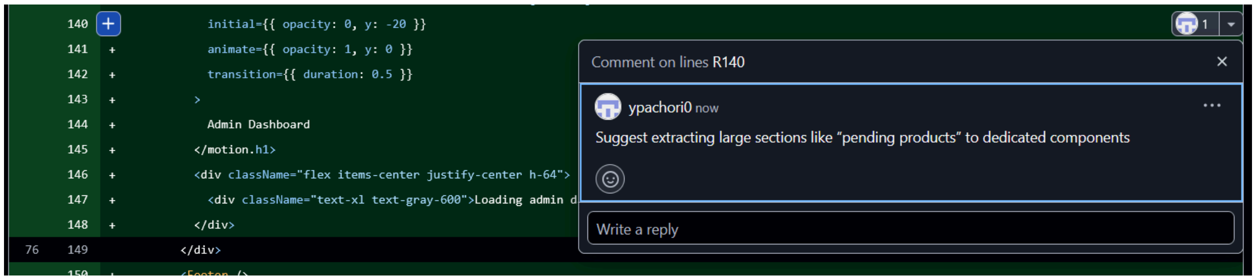
Review Comments Summary:

Area	**Observation**	**Suggestion**
Navigation & Access	Correctly restricts access to admins using localStorage role checks	Consider using context-based auth protection to centralize logic
API Integration	Endpoints for products, users, and stats are properly consumed using Axios	Ensure proper error boundaries in case any one of the three fails
UI/UX	Excellent use of Lucide icons and tab system for admin views	Suggest adding pagination for user list in "User Management" tab

Code Structure	Large file with conditional render logic for each tab	Break down into subcomponents like <PendingProducts />, <UserTable /> etc.
State Management	useState and useEffect hooks used correctly for fetching and managing data	Good use of centralized fetchAdminData() function

GitHub Inline Review Highlights:

- Line 140: Suggest extracting large sections like "pending products" to dedicated components



```
    140 +          initial={{ opacity: 0, y: -20 }}
    141 +          animate={{ opacity: 1, y: 0 }}
    142 +          transition={{ duration: 0.5 }}
    143 +        >
    144 +          Admin Dashboard
    145 +        </motion.h1>
    146 +        <div className="flex items-center justify-center h-64">
    147 +          <div className="text-xl text-gray-600">Loading admin d
    148 +        </div>
76  149        </div>
```

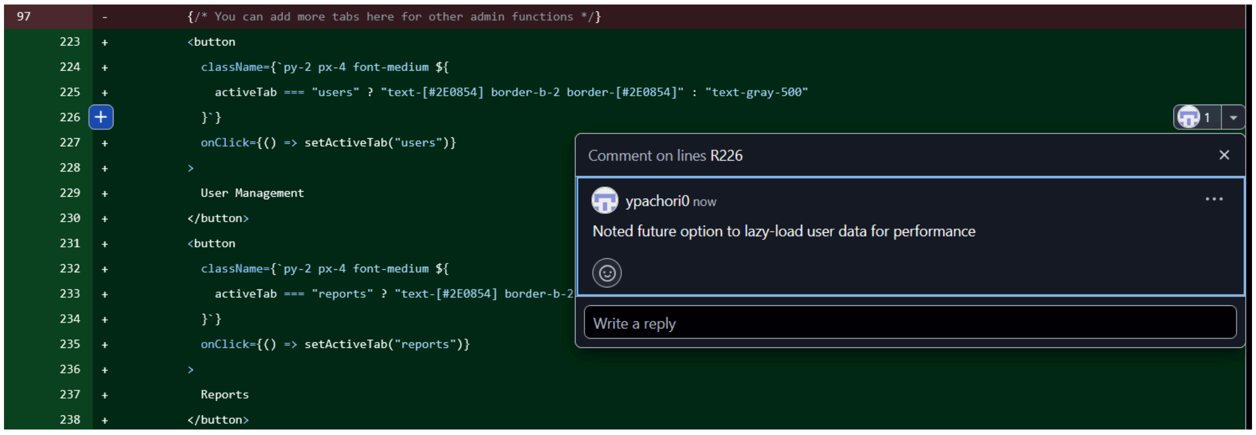
Comment on lines R140

ypachori0 now

Suggest extracting large sections like "pending products" to dedicated components

Write a reply

- Line 226: Noted future option to lazy-load user data for performance



```
97      -        {/* You can add more tabs here for other admin functions */}
    223 +        <button
    224 +          className={`py-2 px-4 font-medium ${
    225 +            activeTab === "users" ? "text-[#2E0854] border-b-2 border-[#2E0854]" : "text-gray-500"
    226 +          }`}
    227 +          onClick={() => setActiveTab("users")}
    228 +        >
    229 +          User Management
    230 +        </button>
    231 +        <button
    232 +          className={`py-2 px-4 font-medium ${
    233 +            activeTab === "reports" ? "text-[#2E0854] border-b-2
    234 +          }`}
    235 +          onClick={() => setActiveTab("reports")}
    236 +        >
    237 +          Reports
    238 +        </button>
```

Comment on lines R226

ypachori0 now

Noted future option to lazy-load user data for performance

Write a reply

GenAI Review of Dev's Code

Prompt Used:

> "Review a React admin dashboard component that displays pending product approvals, user bans, and reports. It uses Axios, React hooks, and Tailwind. Suggest improvements to performance and structure."

Suggestions:

- Recommended splitting into smaller child components
- Suggested adding useCallback or debouncing for API refresh button
- Highlighted good use of loading indicators and toast notifications

Utility Rating: MEDIUM
Benefit: Reinforced modularity and performance optimizations

2. **Peer Review #2: Hsueh (Frontend) Reviewed Daniel's Backend Code (admin.py)**

Code Reviewed:

- File: admin.py
- Feature: Backend routes for admin functionality – moderation, reports, dashboard stats

Review Request:

Daniel shared the feature/admin-api-moderation branch and asked Hsueh to provide a frontend-consumption-focused review.

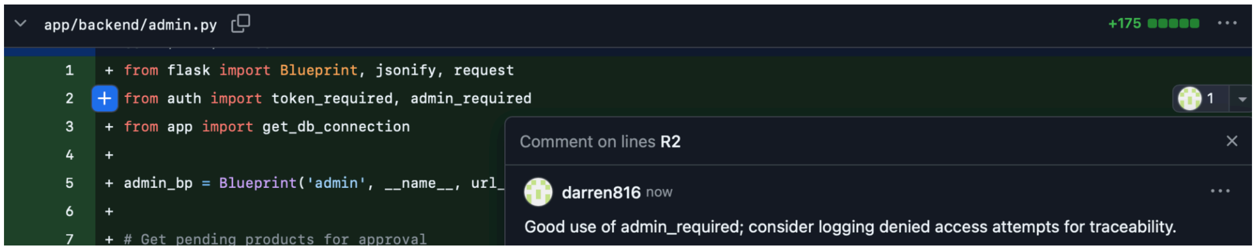
Review Comments Summary:

Area	Observation	Suggestion
Access Control	All routes properly protected with @admin_required	Very secure — just make sure decorators log denied access

API Design	Endpoint structure is RESTful and clearly named	Suggest clearer status codes in failure cases (e.g., 404 if report not found)
DB Usage	Queries are parameterized and avoid injection risks	Could use with conn.cursor() for better resource management
Reuse & Modularity	Repeated status validation in multiple functions	Suggested a helper function for status validation
Testability	Return data is JSON and frontend-compatible	Well-aligned with frontend needs for list display and error messages
API Usage	Limit is not capped	Add a reasonable upper limit (e.g., min(limit, 500))
Data Aggregation	Dashboard queries are clean and efficient	Good use of SQL aggregation to support frontend

GitHub Inline Review Highlights:

- Line 2: Good use of admin_required; consider logging denied access attempts for traceability.



app/backend/admin.py +175

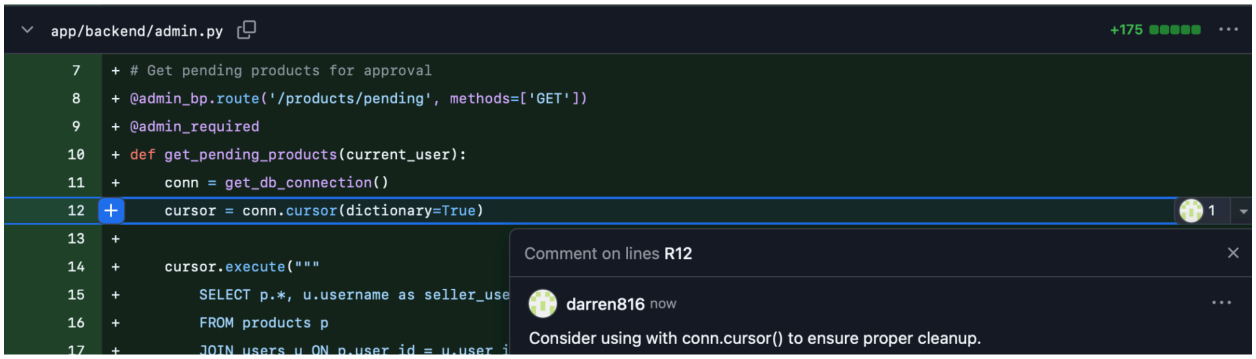
```
1  + from flask import Blueprint, jsonify, request
2  + from auth import token_required, admin_required
3  + from app import get_db_connection
4  +
5  + admin_bp = Blueprint('admin', __name__, url_
6  +
7  + # Get pending products for approval
```

Comment on lines R2

darren816 now

Good use of admin_required; consider logging denied access attempts for traceability.

- Line 12: Consider using with conn.cursor() to ensure proper cleanup.



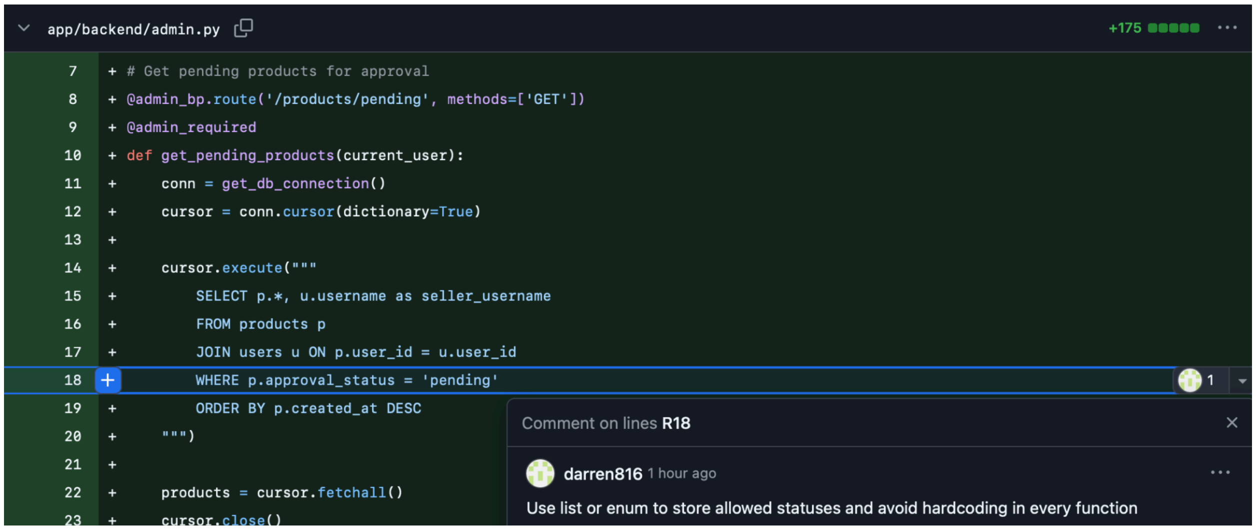
```
app/backend/admin.py                                                   +175
 7 + # Get pending products for approval
 8 + @admin_bp.route('/products/pending', methods=['GET'])
 9 + @admin_required
10 + def get_pending_products(current_user):
11 +     conn = get_db_connection()
12 +     cursor = conn.cursor(dictionary=True)                          1
13 +
14 +     cursor.execute("""
15 +         SELECT p.*, u.username as seller_use
16 +         FROM products p
17 +         JOIN users u ON p.user_id = u.user_i
```

Comment on lines **R12**

darren816 now

Consider using with conn.cursor() to ensure proper cleanup.

- Line 18: Use list or enum to store allowed statuses and avoid hardcoding in every function



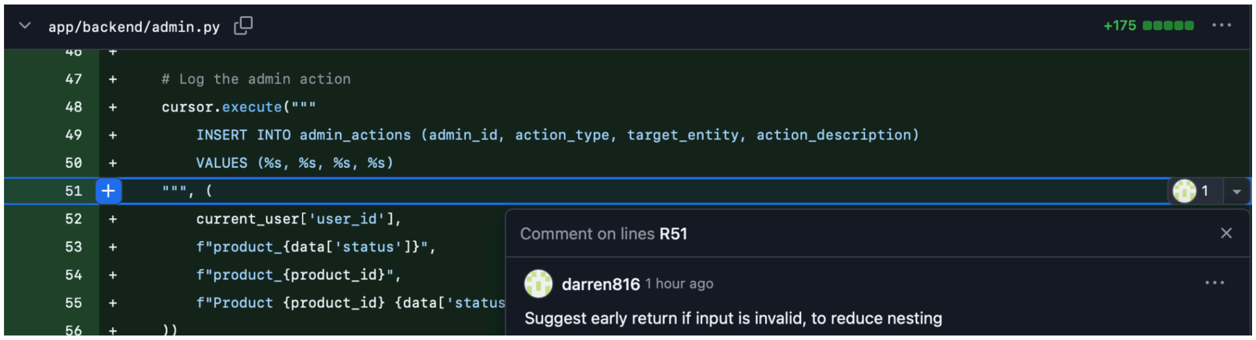
```
app/backend/admin.py                                                   +175
 7 + # Get pending products for approval
 8 + @admin_bp.route('/products/pending', methods=['GET'])
 9 + @admin_required
10 + def get_pending_products(current_user):
11 +     conn = get_db_connection()
12 +     cursor = conn.cursor(dictionary=True)
13 +
14 +     cursor.execute("""
15 +         SELECT p.*, u.username as seller_username
16 +         FROM products p
17 +         JOIN users u ON p.user_id = u.user_id
18 +         WHERE p.approval_status = 'pending'                        1
19 +         ORDER BY p.created_at DESC
20 +     """)
21 +
22 +     products = cursor.fetchall()
23 +     cursor.close()
```

Comment on lines **R18**

darren816 1 hour ago

Use list or enum to store allowed statuses and avoid hardcoding in every function

- Line 51: Suggest early return if input is invalid, to reduce nesting



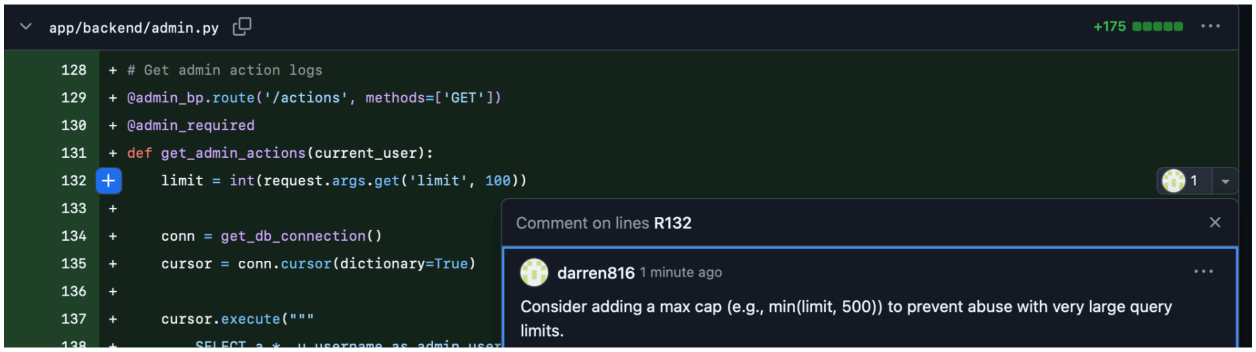
```
app/backend/admin.py                                                   +175
46 +
47 +     # Log the admin action
48 +     cursor.execute("""
49 +         INSERT INTO admin_actions (admin_id, action_type, target_entity, action_description)
50 +         VALUES (%s, %s, %s, %s)
51 +     """, (                                                         1
52 +         current_user['user_id'],
53 +         f"product_{data['status']}",
54 +         f"product_{product_id}",
55 +         f"Product {product_id} {data['status
56 +     ))
```

Comment on lines **R51**

darren816 1 hour ago

Suggest early return if input is invalid, to reduce nesting

- Line 132: Consider adding a max cap (e.g., min(limit, 500)) to prevent abuse with very large query limits.



app/backend/admin.py +175

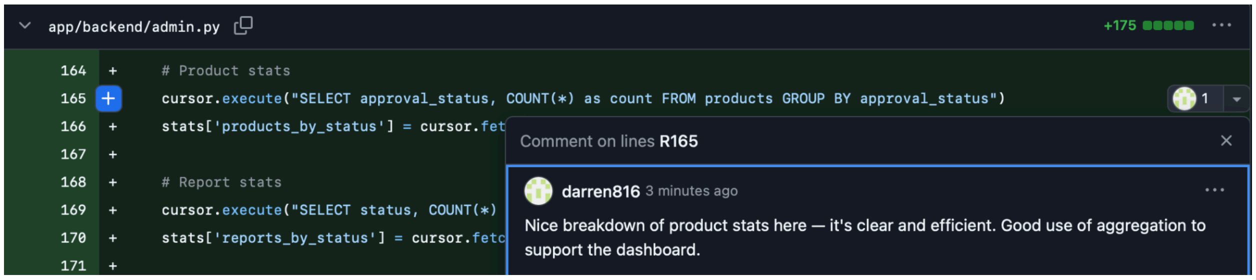
```
128 + # Get admin action logs
129 + @admin_bp.route('/actions', methods=['GET'])
130 + @admin_required
131 + def get_admin_actions(current_user):
132 +     limit = int(request.args.get('limit', 100))
133 +
134 +     conn = get_db_connection()
135 +     cursor = conn.cursor(dictionary=True)
136 +
137 +     cursor.execute("""
138 +         SELECT a.*, u.username as admin_user
```

Comment on lines **R132**

darren816 1 minute ago

Consider adding a max cap (e.g., min(limit, 500)) to prevent abuse with very large query limits.

- Line 165: Nice breakdown of product stats here — it's clear and efficient. Good use of aggregation to support the dashboard.



app/backend/admin.py +175

```
164 +     # Product stats
165 +     cursor.execute("SELECT approval_status, COUNT(*) as count FROM products GROUP BY approval_status")
166 +     stats['products_by_status'] = cursor.fet
167 +
168 +     # Report stats
169 +     cursor.execute("SELECT status, COUNT(*)
170 +     stats['reports_by_status'] = cursor.fetc
171 +
```

Comment on lines **R165**

darren816 3 minutes ago

Nice breakdown of product stats here — it's clear and efficient. Good use of aggregation to support the dashboard.

GenAI Review of Daniel's Code

Prompt Used:

> "Review this Flask Blueprint backend code for an admin panel with endpoints to approve/reject products, view reports, and get dashboard stats. Suggest improvements in performance and structure."

Suggestions:

- Use context managers (with conn.cursor() as cursor) for DB operations
- Consolidate repeated status validation
- Consider separating out logging logic for admin actions

Utility Rating: HIGH
Benefit: Helped emphasize DRY principles and secure DB patterns

5) Security Self Check

This section summarizes our security strategy for protecting the key assets of **Gator Market**, covering both backend (Flask, MySQL) and frontend (React) implementations. We have followed OWASP top 10 recommendations, secure database design, and enforced strict input validation and role-based access control.

Asset to be Protected	Types of Possible/Expected Attacks	Consequence of Security Breach	Your Strategy to Mitigate/Protect the Asset
User credentials (email, password)	Brute-force login, credential stuffing, data breach	Account hijacking, identity theft	Passwords are hashed using bcrypt before storage. Login checks include password hash validation. No plaintext passwords are stored. JWT tokens used for session management.
SFSU email-only registration	Fake user creation using non-SFSU emails	Unauthorized access to student-only features	Regex-based validation (@sfsu.edu enforced) and domain matching on backend during registration (auth.py)
User input in forms	SQL injection, XSS (Cross-Site Scripting)	Data compromise, stored XSS attacks	Backend uses parameterized SQL queries with placeholders (e.g., %s). Client-side and backend validations restrict script inputs.

Product listings	Spam, inappropriate content	Loss of trust, abuse of platform	Admin approval is required before listings go public. Admins can review/reject content.
In-app messaging	Injection, phishing attempts	Abuse of communication channel, potential scams	Input sanitization in message content, limit on message size. No file/image sending permitted.
Bookmarked/wishlist products	JSON injection in MySQL JSON fields	Logic corruption or data overwrite	All JSON operations are server-controlled using JSON_APPEND, JSON_SEARCH. Inputs are strictly type-checked before database operations.
Listing reports / user reports	Report spam, bypass moderation	Admin overwhelmed with invalid reports	Only authenticated users can report. Cannot self-report. Backend enforces logic and logs all actions.
Admin endpoints	Unauthorized access, privilege escalation	Platform compromise	All admin routes protected by @admin_required decorator which checks role before access is granted.

Password Storage Strategy

- Passwords are hashed with bcrypt (12 rounds) before storing in the users table.
- We never store plaintext passwords.
- Password complexity is enforced via regex: minimum 8 characters, uppercase, lowercase, number, and symbol.

Input Validation (Implemented in auth.py and Frontend Forms)

Field	Validation	Regex/Check Used
Username	4–20 chars, alphanumeric and underscore	^[a-zA-Z0-9_]{4,20}$
Email	Must end in @sfsu.edu	^[a-zA-Z0-9._-]+@sfsu\.edu$
Password	Min 8 chars, uppercase, lowercase, digit, symbol	^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,64}$
Search Input	Up to 40 alphanumeric characters	^[a-zA-Z0-9\s]{0,40}$
First/Last Name	2–30 alphabetic characters	^[a-zA-Z\s'-]{2,30}$

Terms Acceptance	Required checkbox on registration form	Checked before submission

Additional Protections

- **JWT-based Authentication**: Used for login and route protection. Tokens expire after 24 hours.
- **CORS Policy**: Strict CORS settings in app.py to allow only frontend domain (https://csc648g1.me).
- **Rate Limiting (Planned)**: Admin and login endpoints are planned to be protected using Flask-Limiter before final deployment.
- **S3 Upload Security**: All file uploads are routed through pre-signed URLs with limited access and expiration.
- **Role-Based Access**: Admin routes protected using admin_required() decorator, which wraps token_required() and checks user role.

6) Non-Functional Requirements Self-Check

#	Non-Functional Requirement	Status	Comments
1	Application shall be developed, tested and deployed using tools and cloud servers approved by Class CTO	DONE	AWS EC2, MySQL, and Flask used
2	Application shall be optimized for standard desktop/laptop browsers (latest 2 versions of 2 major browsers)	DONE	Chrome tested, Firefox tested
3	Application functions shall render well on mobile devices (no native app required)	DONE	Responsive layout implemented
4	Posting of sales info and messaging shall be limited only to SFSU students	DONE	Email must contain "@sfsu.edu"
5	Critical data shall be stored in the database on the team's deployment server	DONE	All user and product data stored in AWS RDS (MySQL) tied to EC2 deployment
6	No more than 50 concurrent users shall be accessing the application at any time	DONE	To ensure compliance with the performance requirement of no more than 50 concurrent users (Requirement #6), we

			configured NGINX with a connection limit directive using the limit_conn module. This was added to the NGINX configuration file (nginx.conf) as follows: nginx CopyEdit limit_conn_zone $binary_remote_addr zone=addr:10m; limit_conn addr 50;
7	Privacy of users shall be protected	DONE	User data is protected through token-based authentication, encrypted passwords (bcrypt), and restricted access to sensitive endpoints.

8	The language used shall be English	DONE	Entire interface and content in English
9	Application shall be very easy to use and intuitive	DONE	Simple and clean UI with guided forms
10	Application shall follow established architecture patterns	DONE	MVC + RESTful API structure
11	Application code and its repository shall be easy to inspect and maintain	DONE	Codebase organized using clear folder structure, naming conventions, and documented routes; version control maintained via GitHub with meaningful commits.
12	Google Analytics shall be used	DONE	Google Analytics tracking code embedded in frontend for usage insights
13	No e-mail/chat services allowed; only internal messaging with single round allowed	DONE	In-app message system implemented

14	No pay functionality or payment simulation allowed in UI	DONE	No payment logic or UI placeholder present
15	Site security: basic best practices must be applied (password encryption, input validation, etc.)	DONE	Passwords hashed with bcrypt; input validation implemented on client and server side
16	Media formats must be current market standard (e.g., jpg, png)	DONE	jpg/png used for product images
17	Modern SE processes/tools shall be used, including collaborative and GenAI-assisted development	DONE	GitHub, Trello, ChatGPT used throughout
18	UI must display: "SFSU Software Engineering Project CSC 648-848, Spring 2025. For Demonstration Only" on every page header	DONE	Required disclaimer text added to all page headers via shared layout component

7) Use of GenAI Tools Summary

Throughout the development of Milestone 4, our team extensively used **ChatGPT (GPT-4)** to enhance clarity, speed, and consistency across technical and design deliverables. This builds on our prior experiences in Milestone 1 and 2, where GenAI tools were used to support brainstorming, functional breakdowns, and database planning.

Tools Used

- ChatGPT GPT-4 (OpenAI)

Where We Used GenAI in Milestone 4

- **Usability Test Plan:** Helped draft clear test objectives, step-by-step task instructions, and Likert-scale questions.
- **QA Test Plan:** Improved test case structure and input/output descriptions; suggested broader test scenarios.
- **Code Review:** Used to analyze React component logic for clarity, naming conventions, and missing error handling.
- **Writing Assistance:** Ensured professional and consistent tone throughout the milestone document.

Key Example Prompts

- *“Help me write a usability test plan for a product upload feature that includes image upload and safety options.”*
- *“Generate QA test cases for search and filter functions on an online marketplace.”*
- *“Review this JavaScript code and suggest improvements in naming, security, and structure.”*

Benefits Observed

- Accelerated writing and reduced rework cycles
- Improved clarity in test plans and documentation
- Provided second opinions on code structure and possible edge cases

Limitations

- Needed human validation for contextual accuracy
- Occasionally offered too generic suggestions for SFSU-specific constraints

GenAI Utility Ranking: HIGH

Overall, GenAI played a critical supporting role across planning, testing, and review tasks in M4. While it did not replace our work, it significantly enhanced the speed and polish of our output. We expect to continue using GenAI tools for Milestone 5 in areas such as user documentation, final QA refinement, and presentation design.