2025/2026 SENIOR DESIGN DESIGN DOCUMENT

Ames After Dark

Group Number: sdmay26-42
Client: Chase Anderson
Advisor: Professor Maruf
Team Email: sdmay26-42
Team Website:
https://sdmay26-42.sd.ece.iastate.edu/

Chase Anderson: Quality Control Lead (Full Stack)
Jaya Davis: Team Communications Lead (Front End)
Nathan Couture: DevOps Lead (Back End)
Nathan Krieger: Client Liaison (Full Stack)
Analyn Seeman: Project Management (SCRUM Master) (Front End)
Maggie Sullivan: Documentation Lead (Full Stack)
Geni Williams: Meeting Scribe (Full Stack)

EXECUTIVE SUMMARY

Ames After Dark is a mobile application designed to transform the nightlife experience in Ames, Iowa, by creating a centralized, real-time hub for events, bar specials, social connection, and safety. Today, students and residents rely on scattered sources like social media posts, static websites, or word-of-mouth to decide where to go out, which often results in inefficiency, missed opportunities, and safety concerns when groups get separated. This project addresses those problems by integrating event discovery, mapping, photo sharing, and location tracking into a single cross-platform mobile app.

Users will be able to quickly browse a feed of nightly events and specials, check live highlights such as the busiest bar or cheapest drinks, and apply filters to narrow their choices. An interactive map will display all Ames bars with their current open/closed status, highlight user favorites, and allow friends to share locations or plan bar crawls together. Beyond navigation, the app will provide weekly photo galleries from professional photographers, allowing students to relive and share memories, and a searchable bar directory with menus, deals, hours, and booking capabilities for events like graduations or private parties. Safety is a core consideration, with features like mutual friend location sharing, one-tap check-ins, and notification settings designed to promote responsible nightlife.

Bar owners will also benefit through a dedicated admin portal where they can upload deals, events, photos, and manage profiles, supported by role-based permissions and potential analytics in the future. The system will be built with modern, scalable technologies-React Native for the frontend, Node.js and PostgreSQL for the backend, Firebase for secure authentication, and Expo Location & React Native Maps for geolocation-hosted on cloud free-tier services during development. While the scope of the senior design project is a working prototype rather than a production release, the deliverable will demonstrate all core features, validate technical feasibility, and provide a compelling proof-of-concept. Ultimately, Ames After Dark balances user needs for convenience, safety, and connection with bar owners' needs for visibility and engagement, offering a solution with real-world impact and strong portfolio value for the development team.

LEARNING SUMMARY

Development Standards & Practices Used

- Followed software engineering best practices including modular design, version control (GitHub), agile development, and iterative testing.
- Applied UI/UX standards for mobile applications: accessibility (contrast, text size, alt-text), and responsive layouts.
- Used security practices such as secure authentication, data validation, and role-based access for bar owners.
- Backend hosted with Node.js with our database on AWS, real-time syncing, and API-first architecture.

Summary of Requirements

- **Must-Haves (MVP):** Navigation bar (5 tabs: Home, Map, Gallery, Bars, Account), real-time event/specials feed, interactive bar map with friend location sharing (restrictions implemented based on time and location), weekly photo galleries, bar directory (menus, deals, hours), user accounts, admin portal for bars, secure authentication, and cloud-hosted backend.
- **Should-Haves:** Push notifications, group crawl tracking, verified photographer uploads, advanced booking, notification preferences, admin analytics.
- **Could-Haves:** Trending events (based on location popularity in real time), loyalty rewards/gamification, real-time wait times or cover charges, safe-route home links.
- **Won't-Haves:** In-app ticket sales, in-app payments, public map visibility of all users, unlimited user-generated uploads, direct messaging between users.

Applicable Courses from Iowa State University Curriculum

- **COM S 309 - Software Development Practices:** agile methods, GitHub version control, full-stack teamwork
- **SE 319 - Construction of User Interfaces:** mobile-first UI/UX design, usability, accessibility

- **COM S 363 - Introduction to Databases:** relational schema design, SQL queries, data modeling
- **CPR E 308 - Operating Systems/Networking:** backend client-server interactions, cloud hosting considerations
- **SE 329 - Software Project Management:** project scoping, requirement prioritization, deliverables
- **SE 417 - Software Testing:** unit testing, integration testing, validation planning
- **CYB E 230 / CYB E 231:** cyber security fundamentals, attack prevention tools, and network security methods

New Skills/Knowledge Acquired that was not taught in Courses

- **Mobile cross-platform development** with React Native.
- **Real-world cloud deployment practices** using Firebase/Supabase and evaluating scalability on AWS.
- **Integration of third-party APIs** (Maps, Firebase Authentication) and handling geolocation.
- **Admin portal design** for non-technical users, including CRUD tools and role-based permissions.
- **Practical UI prototyping** in Figma and translation into functional mobile code.
- **Domain-specific knowledge** of bar/nightlife operations in Ames, including specials, events, and booking workflows.

TABLE OF CONTENTS

LIST OF FIGURES

No figures are included in the document yet.

1. INTRODUCTION

1.0 Problem Statement

Nightlife in Ames, Iowa, plays a big role in student and community life, but finding accurate, up-to-date information about what is happening is harder than it should be. Students often learn about bar specials, live music, or events through scattered Instagram posts, flyers, or word-of-mouth. This makes it easy to miss out on opportunities, wastes time trying to decide where to go, and sometimes creates safety issues when groups get separated late at night. There is no single platform that brings all of this information together while also supporting the social and safety needs of the people who participate in Ames nightlife. Ames After Dark seeks to address this gap by offering a one-stop mobile app where users can discover real-time events and deals, connect with friends on an interactive map, and stay safe while enjoying the nightlife scene. By creating a more connected and informed experience, the app not only makes going out easier and more fun but also supports safety and community in Ames.

1.1 Intended Users

The primary users of Ames After Dark are college students at Iowa State University and nearby schools. These students are highly social, often go out in groups, and want quick, reliable ways to find the best events or deals on any given night. Their needs include convenience (finding specials fast), connection (staying in touch with friends on bar crawls), and safety (knowing where their group is and having quick access to check-ins).

A second key group is Ames residents who enjoy nightlife but are not students. These users may be young professionals, alumni, or locals who want to explore the bar scene without relying on word-of-mouth. Their needs are slightly different: they may prioritize reliable information about events, bar hours, or booking spaces for special occasions like birthdays or graduations.

Finally, bar owners and event organizers are also important users. They benefit by having a direct way to share deals, events, and photos with a targeted audience. Their needs include easy-to-use tools to update specials, manage bar profiles, and attract more customers. By giving bars a centralized platform, Ames After Dark not

only supports their business goals but also ensures that the information students and residents see is timely and accurate.

Together, these groups highlight the dual value of the app: making nightlife in Ames more fun and safer for users, while also strengthening the connection between bars and their community.

2. REQUIREMENTS, CONSTRAINTS, AND STANDARDS

2.1 Requirements & Constraints

2.1.1 Functional Requirements

Functional requirements define the essential behaviors and capabilities of the system. These requirements are directly tied to user needs such as event discovery, social connection, and nightlife safety.

- Provide a real-time feed of nightly events and drink specials.
- Include an interactive map of Ames bars with open/closed status and friend locations (mutual, opt-in).
- Support a searchable bar directory with menus, specials, and hours, with the ability to favorite/unfavorite bars.
- Provide weekly photo galleries uploaded by verified photographers, organized by bar and date, with download/share functionality.
- Support secure user registration, login, profile management, and friend connections.
- Provide an admin portal where bar owners can upload deals, events, and photos, with role-based permissions.
- Maintain cloud-hosted backend and relational database with real-time syncing.

2.1.2 User Experience Requirements

User experience requirements describe qualities that affect how users interact with the system, focusing on convenience, usability, and safety.

- Update bar open/closed status at least once per hour.
- Provide filtering options for events, specials, and bar availability.

- Restrict location sharing to mutual friends and explicit opt-in.
- Allow users to browse and filter photo galleries by bar and date.
- Enable one-tap interactions for common actions (e.g., checking in, toggling location visibility).

2.1.3 Physical / Resource Requirements

These requirements specify the platforms, infrastructure, and resources needed to implement and operate the system.

- Must run on both iOS and Android platforms.
- Backend hosted on cloud services using free-tier plans during development.
- Use a relational database (e.g., PostgreSQL) to store bar data, user accounts, and event information.
- Integrate with third-party APIs for maps and authentication.

2.1.4 Aesthetic / UI Requirements

Aesthetic and UI requirements ensure the interface is visually clear, consistent, and usable in fast-paced nightlife environments.

- Use a bottom navigation bar with five core sections: Home, Map, Gallery, Bars, and Account.
- Fonts and color contrasts must follow accessibility guidelines (e.g., WCAG minimum contrast ratios).
- Visual layout should prioritize quick, one-handed interactions and readability in low-light environments.
- Use consistent iconography and theming across all screens.

2.1.5 Constraints

Constraints represent boundaries and limitations that affect design decisions.

- The system will not include in-app ticket sales or payments.
- User location visibility restricted to mutual friends only; no public tracking.
- Initial events and deal data may be manually curated rather than fully automated.
- Development is limited to the senior design two-semester timeframe; production-level scaling is not feasible.
- Must rely on free-tier cloud hosting during senior design development.

2.2 Engineering Standards

Engineering standards are important in ensuring quality, interoperability, and usability in software projects. For Ames After Dark, applying the right standards helps the team avoid vague requirements, design accessible interfaces, and ensure reliable system behavior. The following standards were selected as most relevant to our project:

2.2.1 ISO/IEC/IEEE 29148:2018 (Systems and software engineering - Life cycle processes - Requirements engineering)

- **Description**: This standard provides a framework for effective requirements engineering across the system life cycle. It outlines how to write high quality requirements, ensure they are consistent and verifiable, and manage them as projects evolve. It also defines attributes of good requirements, such as being clear, feasible, and testable.
- **Relevance**: For Ames After Dark, ISO/IEC/IEEE 29148:2018 helps the team create precise and measurable requirements. Instead of vague goals like "the app should be fast," this standard guides us to specify requirements such as "bar status must update within one minute." Using this approach improves clarity for developers, ensures testability, and reduces the risk of misunderstandings during design.

2.2.2 ISO/IEC 27001:2022 (Information security, cybersecurity and privacy protection - Information security management systems - Requirements)

- **Description**: This standard defines requirements for establishing, implementing, and continually improving an Information Security Management System (ISMS). It aims to protect confidentiality, integrity, and availability of data and includes controls such as access control, encryption, auditing, and secure development.
- **Relevance**: Because Ames After Dark handles user accounts, location sharing, and possibly sensitive user data, ISO/IEC 27001:2022 helps define how we protect that data. For example, we'd use access controls, encryption of sensitive data, and more to protect all sensitive data. Applying this standard helps us better defend against breaches and build trust with users.

2.2.3 ISO/IEC 25010:2023 (Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model)

- **Description**: This standard defines a comprehensive model for evaluating the quality of Information and Communication Technology (ICT) products and software. It identifies nine characteristics including functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety. The standard provides a reference model for specifying, measuring, and evaluating product quality throughout its lifecycle, addressing how well a product meets each quality attribute.
- **Relevance**: For Ames After Dark, ISO/IEC 25010:2023 provides a structured framework for defining and evaluating software quality across multiple dimensions. By referencing this model, the team can set clear targets for performance efficiency (real-time event updates), reliability (consistent uptime), and interaction capability (intuitive mobile navigation). The inclusion of safety and security aspects also supports responsible handling of user location and data. Applying this standard ensures that the app not only functions correctly but also meet high expectations for quality, usability, and long-term maintainability.

3. PROJECT PLAN

3.1 Project Management/Tracking Procedures

Our team is adopting an Agile development approach, emphasizing iterative progress, flexibility, and regular communication. Because Ames After Dark is a complex application involving both frontend and backend development, Agile allows the team to deliver functionality in manageable increments, respond to feedback, and adapt design decisions as new requirements emerge. Each sprint will focus on completing a subset of prioritized features and refining them through testing and review.

To track progress and coordinate effectively, the team uses the following tools and methods:

- Gitlab - used for version control, issue tracking, and merge request management.
- Microsoft Teams - used for file sharing, meeting information, and longer technical messages and questions.
- Group SMS Chat - used for fast updates, reminders, and coordinating meeting times or small decisions.
- Face-to-Face Meetings - used during scheduled team work sessions to discuss major updates, plan tasks, and resolve technical or design issues collaboratively.
- Email - used to communicate formally with the advisor, particularly for scheduling and milestone updates.

The combination of these tools supports transparency, accountability, and continuous progress tracking throughout both semesters.

3.2 Task Decomposition

The Ames After Dark system is divided into several high-level components, each with associated tasks and subtasks. This decomposition supports parallel development and clearer ownership of deliverables.

1. Project Setup and Planning
    a. Define requirements and constraints
    b. Select technology stack (frontend, backend, hosting, APIs)
    c. Create development repositories and documentation structure
2. Frontend Development
    a. Design and implement navigation bar and screen layouts
    b. Develop core screens (Home, Map, Gallery, Bars, Friends)
    c. Implement photo gallery view with filters
    d. Integrate authentication and location permissions
3. Backend Development
    a. Design and implement database schema
    b. Develop APIs for events, bar data, and authentication
    c. Configure Firebase for secure login and storage
    d. Manage real-time syncing
4. Integration and Testing
    a. Connect frontend with backend APIs
    b. Conduct unit and integration testing
    c. Address performance bottlenecks and security issues
5. Deployment and Documentation

a. Deploy prototype to beta testers/test devices
b. Prepare final documentation and presentation

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

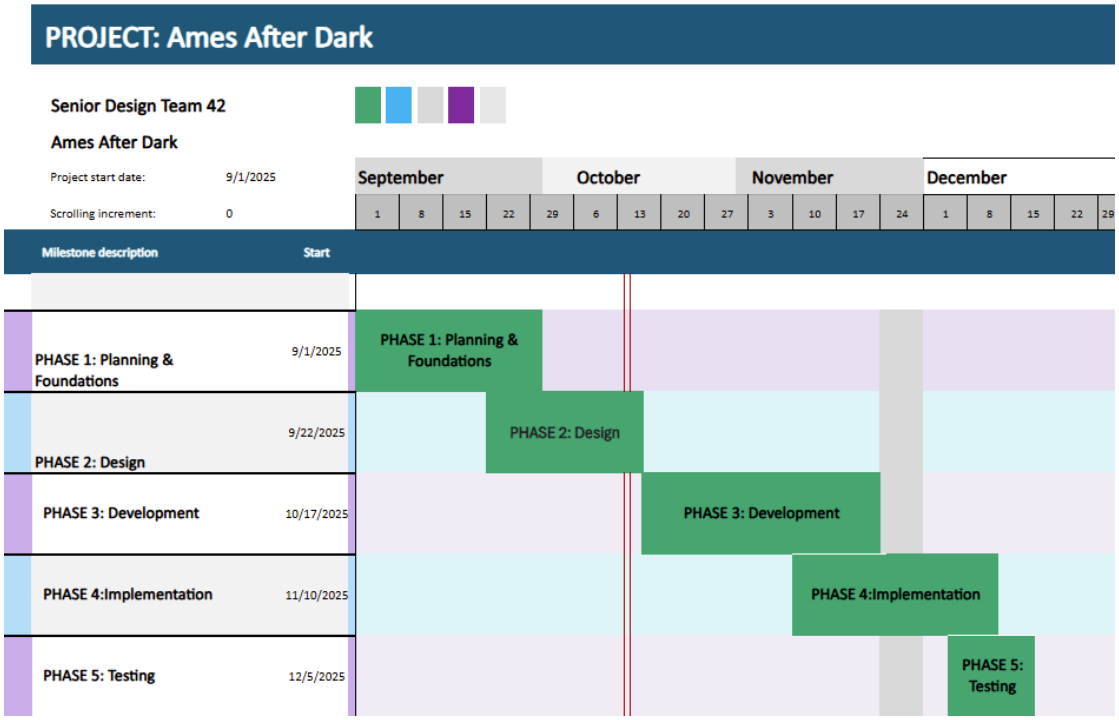
Key milestones are established to measure progress and ensure timely completion. Each milestone corresponds to functional goals or sprint deliverables.

Milestone	Target Completion	Metric / Evaluation Criteria
Screen Finalization	September 2025	Team consensus on finalized UI layout and navigation flow in Figma; approval from advisor and client.
ER Diagram & Database Design	October 2025	Logical and physical ERD completed, reviewed, and approved; database schema accurately represents all major entities and relationships.
Design Documentation Completion	October 2025	Design document sections drafted, internally reviewed, and integrated with minor revisions.
Technology Stack Confirmation	October 2025	Full team agreement on frameworks, APIs, and hosting platforms; compatibility validated on all developer machines.
Project Requirements Lock	October 2025	Functional and non-functional requirements finalized, reviewed by team, and approved by client.
Frontend Structure Implemented	November 2025	Navigation and five core screens (Tonight, Map, Gallery, Bars, Account) operational in Expo simulator.
Backend API and Database Operational	November 2025	CRUD operations for users, bars, events, and photos verified through Postman; live connection to Supabase confirmed.
Integration Between Frontend and Backend	December 2025	Data retrieved and displayed in app without major errors; successful end-to-end tests for login, bar listings, and photo gallery.
User Testing Round 1 - Internal Team	February 2026	Internal test plan executed; >90 % of major functionality verified by development team; bugs logged and prioritized.

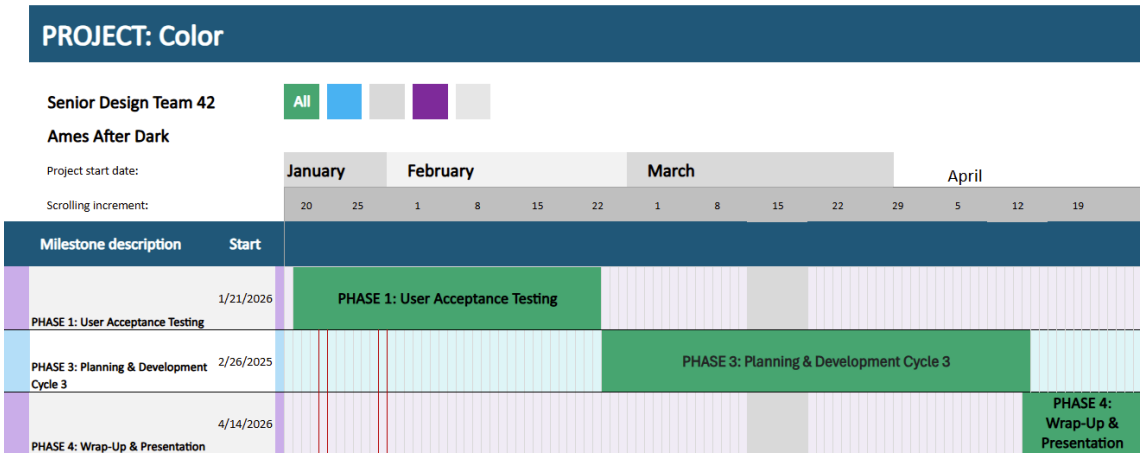
User Testing Round 2 - Controlled Beta Group	March 2026	10-15 student testers complete structured usability session; feedback aggregated and ≥80 % satisfaction rating achieved.
Advanced Feature Integration	March 2026	Friend map updates, real-time specials, and admin uploads implemented; regression tests pass with <5 % critical failures.
User Testing Round 3 - Live Bar Field Testing	April 2026	Successful pilot with 2-3 local bars; patrons can view live data and photos; app stability ≥95 % uptime during live use.
Final Fixes and Polish	Mid-April 2026	All critical bugs resolved; UI consistent with design standards; verified through advisor review.
Final Documentation and Presentation	Late April 2026	Technical report, user guide, and poster complete; live demonstration to faculty with all MVP features fully functional.

3.4 Project Timeline/Schedule

Semester 1



Semester 2



Detailed Schedules

This is a summary of the project timeline altered for readability. View the full Gantt chart here:

- Full Gantt Charts

3.5 Risks and Risk Management/Mitigation

Risk	Probability	Mitigation Strategy
Integration issues between frontend and backend	0.6	Conduct early API testing; maintain consistent data structures across team
API rate limits or authentication errors	0.4	Implement caching and fallback data for critical features
Schedule delays due to exams/holidays	0.4	Plan lighter sprints around academic breaks
Security vulnerabilities (data leaks or poor access control)	0.3	Follow ISO/IEC 27001 principles, apply secure login and encryption
Cloud service limits (free-tier restrictions)	0.5	Optimize data storage and monitor usage; migrate if necessary

3.6 Personnel Effort Requirements

The table below estimates the total person-hours required for major tasks in the Ames After Dark project. Estimates are based on relative task complexity, expected learning curves, and each member's role specialization. These numbers reflect projected effort across both semesters of senior design.

Task	Team Members	Estimate Hours	Description

Project planning and documentation	All	60	Define project scope, requirements, constraints, and maintain documentation.
Frontend Development	Jaya, Analyn, Maggie, Chase, Nathan K, Nate C	160	Develop main app screens, implement navigation, and integrate visual components with backend APIs.
Backend Development	Geni, Nate C, Chase, Nathan K, Maggie	170	Set up database schema, implement API endpoints, and establish data synchronization.
Integration and Testing	All	100	Perform unit, integration, and user testing; address bugs and verify system functionality across platforms.
Deployment and Hosting Setup	Nate C, Geni	40	Configure cloud hosting and app distribution for testing
Final Presentation and Deliverables	All	50	Prepare final demo, documentation, and presentation materials for faculty.

3.7 Other Resource Requirements

Aside from personnel effort and time, the Ames After Dark project relies on several external and technical resources necessary for successful completion.

Resource	Type	Purpose/Description
Bar Participation	External Partner	Collaboration with local Ames bars to gather accurate event, deal, and profile data for testing.
Photographer Participation	External Partner	Work with bar photographers to source sample gallery images and verify the upload and tagging process.
Cloud Hosting	Technical	Provide authentication, database storage, and real-time synchronization for user and bar data.
Expo Location and React Native Maps	Technical	Enable location-based services including friend sharing and bar mapping.
Development Tools	Technical	Includes Visual Studio Code, GitLab, Figma, and Teams for development, version control, design, and communication.
Mobile Devices (iOS/Android)	Hardware	Used for testing app performance and interface compatibility across platforms.
University Resources	Institutional	Access to ISU's network, software licenses, and

		senior design facilities for collaboration and testing.

These resources ensure the team can simulate realistic use cases, meet accessibility and reliability goals, and deliver a functional prototype by the end of the project timeline.

4. DESIGN

4.1 Design Context

4.1.1 Broader Context

Ames After Dark is designed to enhance nightlife experiences in Ames, Iowa, by connecting students, residents, and bar owners through a single, real-time mobile platform. It supports safer, more informed social experiences while helping local businesses reach customers efficiently.

Area	Description	Examples
Public health, safety, and welfare	The app promotes safe social behavior by allowing users to share locations only with mutual friends, check in with one tap, and view open/closed statuses of venues in real time.	Encourages group safety and accountability during nightlife activities, reduces the risk of losing contact with friends late at night, make sure friends get home safely
Global, cultural, and social	The app reflects the social culture of a college town while emphasizing inclusivity and responsible community engagement. The app aligns with university and community values of safety, fun, and connection.	Builds a sense of community among students and residents, respects diverse social preferences, encourages the support of local business
Environmental	The system's digital-only design limits physical advertising materials such as flyers and posters, reducing paper waste, and printing energy.	Decreases reliance on non-recyclable promotional materials and encourages more sustainable event promotion practices
Economic	The app supports local economic growth by connecting users to local bars and events while providing bar owners with low-cost advertising tools.	Drives customer traffic to small businesses, improves event visibility, maintains affordability for student users

4.1.2 Prior Work/Solutions

Several existing platforms and academic studies address event discovery, social location sharing, and venue mapping. Below is a review of key prior work, followed by a comparison of strengths/weaknesses and how Ames After Dark differentiates itself.

Existing Systems and Research

- The system Group Event Venue Recommendation (GEVR) uses mobility traces and social relationships to recommend venues for groups of mobile users, achieving over 80% accuracy in predicting meeting locations [1].
- The mobile app OutWithFriendz enables users to organize group events, vote on venues, and schedule outings among friends. Zhang et al. [2] studied user mobility and group decision-making processes in the app's development.
- Eventbrite supports event creation, ticketing, and discovery for a wide range of events. While it provides broad coverage, research and review indicate usability and trust issues in certain contexts [3].

Pros and Cons Summary

System	Pros	Cons
GEVR	Strong group recommendation model, leverages mobility/social data	Specialized to group-venue selection; lacks full bar-specials, directory, or nightlife-specific features
OutWithFriendz	Focused on social groups and event planning	Largely research prototype; limited mapping of venues and real-time venue status
Eventbrite	Large user base, broad event catalog, strong promotion tools	Not tailored to local nightlife social mapping; user reviews note issues with event authenticity and usability

How Ames After Dark Differs

Ames After Dark combines the following elements in a way that is not fully addressed by the prior examples:

- A local-town focus (Ames, Iowa) tailored to student and nightlife communities (addressing user context rather than general event listings).
- Integration of live bar open/closed status, deals, and photo galleries together with social friend-location sharing on an interactive map.
- Emphasis on safe social navigation (mutual opt-in location sharing, check-ins) within a nightlife context rather than generic event discovery.
- Admin portal for local businesses to upload specials and photos, creating a two-sided value for users and bar owners.

While the prior systems offer strong foundations in group planning and event discovery, they either lack specialized nightlife, social mapping, and business participation model, or they operate at a broader geographic scale. Ames After Dark fills this niche by combining social safety, nightlife information, mapping, and business interaction in a focused local context.

4.1.3 Technical Complexity

The Ames After Dark system demonstrates technical complexity across multiple interacting subsystems, with each subsystem employing distinct engineering, scientific, and mathematical principles. Together, these components create a full-stack, data-driven mobile ecosystem that integrates real-time information, social features, and location-based services at a level comparable to commercial social-mapping and nightlife applications.

Core Components and Applied Principles

- **Frontend (Mobile Application):**
    - Built with React Native and Expo for cross-platform development on iOS and Android. This subsystem applies software engineering principles such as modular design, state management, and event-driven programming.
    - Human-computer interaction and accessibility principles guide the interface layout, ensuring usability in low-light and mobile environments.
- **Backend API Gateway:**
    - Developed using Express to manage data flow between the frontend, authentication, and database layers. This subsystem applies concepts from network engineering and systems design, supporting scalability and secure client-server communication.
- **Database Layer:**
    - Implemented with PostgreSQL hosted on an Apache Server. It applies to computer science and data management principles, including relational modeling, foreign key constraints, indexing, and query optimization to ensure fast and reliable data retrieval across large datasets of users, events, and photos.
- **Authentication and Security:**
    - Uses Auth0 Authentication to securely manage user and bar owner accounts. This involves cryptographic principles (token-based authentication, hashing) and access control mechanisms such as role-based authorization and least-privilege enforcement to ensure data privacy and trustworthiness.
- **Mapping & Geolocation Engine:**
    - Integrates Expo Location and React Native Maps to visualize bars and the locations of your friends. This subsystem applies geospatial computing principles, including coordinate geometry for mapping and geolocation.
- **Admin Portal:**
    - A separate web interface for bar owners to upload specials, events, and photos. This component applies information systems and web engineering principles through CRUD operations and data validation connecting users and businesses.

- **Real-Time Synchronization and Notifications:**
    - Supports continuous updates for bar statuses, friend check-ins, and event changes. This leverages distributed systems principles such as asynchronous messaging, event propagation, and concurrency control to maintain consistency across all active devices.
- **Safety and Location Sharing Module:**
    - Implements mutual friend-based location sharing and check-ins. This uses mobile computing concepts such as background services, geofencing, and user-consent-driven privacy design to balance safety and performance.

Challenging Requirements

1. **Cross-Platform Integration:** Achieving feature parity between iOS and Android using a single React Native codebase while maintaining native performance and device-specific capabilities.
2. **Real-Time Data Consistency:** Synchronizing dynamic data (bar status, friend locations, live events) across multiple concurrent users, requiring efficient data caching and transaction logic.
3. **Scalable Cloud Architecture:** Deploying a modular backend capable of handling many simultaneous requests, scalable via containerized deployment on our ETG provided Apache Server.
4. **Privacy and Security Standards:** Implementing secure authentication, encrypted communications, and data governance aligned with industry best practices (Auth0).
5. **Geospatial Data Visualization:** Calculating and displaying spatial relationships (routes, distances, nearby venues) in real time, integrating both coordinate math and dynamic UI rendering.
6. **Two-Sided System Design:** Supporting both end-users (students/residents) and business owners (bars) with separate interfaces, workflows, and database permissions.

Comparison to Industry Standards

The project's architecture and feature set approach the complexity of modern commercial platforms such as Yelp, Eventbrite, and Snapchat's snapmap, which combine geolocation, live data feeds, and social interactivity. By integrating these capabilities in a local, nightlife-specific context with real-time updates, Ames After

Dark meets and exceeds the technical scope expected of professional-grade mobile applications.

4.2 Design Exploration

4.2.1 Design Decisions

Several key design decisions have shaped the structure, usability, and long-term scalability of the Ames After Dark system. Each decision was made through a combination of technical evaluation, user-experience considerations, and alignment with project goals for reliability and local impact.

1. Cross-Platform Development Framework (React Native + Expo)

- **Decision:** The team selected React Native with Expo instead of developing separate native apps for iOS and Android.
- **Rationale:** React Native enables a shared codebase across both platforms, reducing development time and maintenance overhead while providing near-native performance. Expo simplifies device testing, permissions, and deployment.
- **Impact on Success:** This decision ensures faster prototyping and consistency in user experience between iOS and Android devices, which is critical for reaching the broadest audience of Ames students and residents. It also allows the team to focus resources on functionality and design rather than maintaining two separate environments.

2. Backend Architecture (Node.js + Express + PostgreSQL on Apache)

- **Decision:** The system uses a RESTful API built with Node.js and Express, connected to a relational PostgreSQL database hosted on an Apache server.
- **Rationale:** The stack provides high flexibility, robust data relationships, and real-time synchronization capabilities. PostgreSQL's structured schema supports complex queries for bars, events, and users, while Node.js provides efficient asynchronous handling of multiple client requests.
- **Impact on Success:** This architecture allows reliable, low-latency access to bar and event data even under concurrent user loads. It also provides clear modular separation between frontend and backend, facilitating future integration with other systems (e.g., automated data feeds from local businesses).

3. Mapping & Geolocation Integration (Expo Location and React Native Maps)

- **Decision:** The project integrates Expo Location and React Native Maps for displaying bar locations, friend positions, and possibly bar-crawl routes.
- **Rationale:** Expo Location and React Native Maps provide built-in foreground and background tracking, geolocation and geofencing, and device native map integration. These tools support features essential to the app, such as friend tracking, displaying bar locations, and the "pinning" of the user's favorite bar(s). It also easily integrates with other Expo libraries for future expansion.
- **Impact on Success:** This decision enhances usability and reliability for real-time map interaction, ensuring that users can trust the spatial data and easily find their friends scattered between nightlife venues. It also adds visual polish and functional depth comparable to industry apps like Snapchat's Snapmap or Yelp.

4. Mutual Opt-In Location Sharing for Safety

- **Decision:** Location visibility between friends is strictly mutual and requires opt-in consent.
- **Rationale:** This approach prioritizes user privacy and safety, aligning with ethical data-handling standards and modern app security expectations.
- **Impact on Success:** By enforcing mutual consent, Ames After Dark builds user trust and mitigates safety risks associated with passive location tracking-supporting one of the app's core goals: safe social navigation at night.

5. Admin Portal for Local Business Participation

- **Decision:** A separate web-based portal allows bar owners to upload events, specials, and photos directly into the system.
- **Rationale:** Including local businesses as data contributors encourages community adoption and ensures more accurate, current nightlife information. The decision supports a two-sided platform model that benefits both users and businesses.
- **Impact on Success:** This creates a sustainable ecosystem where data freshness is maintained without constant manual input from the development team, enhancing scalability and long-term viability.

These design decisions collectively balance technical feasibility, user experience, and community involvement. Each one directly influences the system's performance, maintainability, and trustworthiness-ensuring that Ames After Dark achieves its goal of delivering a reliable, engaging, and safe nightlife companion for Ames, Iowa.

4.2.2 Ideation

When deciding on the backend framework, we brainstormed several options to find a balance between performance, scalability, and ease of development. We used a collaborative approach like the lotus blossom technique, where we started with "backend framework" as the central idea and expanded outward with potential technologies and their pros and cons. Through this we came up with five main options:

1. **Node.js with Express:** We ultimately chose this option because of its lightweight, event-driven nature, large ecosystem of libraries, and smooth

integration with modern front-end frameworks like React Native. It also allows for fast prototyping and real-time features, which fit our app's needs well.

2. **Django (Python):** We considered Django for its built-in features and strong data management capabilities, but its monolithic structure felt less flexible for our asynchronous and mobile-focused architecture.
3. **Spring Boot (Java):** This was an appealing choice for reliability and enterprise-grade scalability, but it required more setup and was less efficient for the smaller-scale and faster-paced development we aimed for.
4. **Flask (Python):** Flask offered simplicity and control, but it lacked many built-in components, which would have required additional development effort to implement authentication, routing, and middleware.
5. **Ruby on Rails:** Rails provided rapid development and convention-over-configuration benefits, but the team had less experience with Ruby, and we wanted to use a JavaScript-based stack for consistency with our front-end.

Through comparing these options, we found that **Node.js with Express** was the best for our project and our existing skillsets.

4.2.3 Decision-Making and Trade-Off

To better evaluate our backend framework options, we did an analysis where we focused on factors such as development speed, scalability, team familiarity, and performance

We used a weighted decision matrix to make our choices. Each criterion was weighed on a scale of 1-5. It was weighted based on its importance to the app.

Framework	Dev Speed	Scalability	Team Familiarity	Performance	**Total**
Node.js	5	4	5	4	**18**

Django	3	5	3	4	**15**
Spring Boot	2	5	5	2	**14**
Flask	3	3	3	4	**13**
Ruby on Rails	4	4	2	3	**13**

Based on the weighted decision matrix above, Node.js and Express became clear winners. Development speed and team familiarity were two areas that were particularly important to our team.

4.3 Proposed Design

4.3.1 Overview

Our current design utilizes a React-Native front end with a Node.js backend. The front end is where the user-facing software is at, including the screen designs and the logic for user interactions. Using React-Native allows us to write one version of the app for both iOS and android. Not only does this simplify and speed up the development lifecycle, but it also eliminates the need for device specific development experience, which would be challenging to meet in a student team like ours. We are using the framework Expo along with React-Native, since it offers convenient functionality for navigation and app structure. We are using Expo-Router, which allows for easy translation between file structure and actual page structure in the app.

For the backend, we are using Node.js along with Express. Node.js allows us to use JavaScript for our backend, which serves as the recipient of API requests and the middleman between the frontend and database. Express allows for an easier and simpler experience with writing and managing API endpoints.

Users are authenticated for the app with the use of Auth0, an authentication service. Auth0 is used on the frontend and provides a holistic login experience. It is also used on the backend to verify requests made by the frontend. Auth0 is convenient because it eliminates a large amount of work that would otherwise need to be done by our team.

Our backend is hosted on servers owned by the ECPE department. In addition to the backend, our database also resides on these servers. Our database uses PostgreSQL.

4.3.2 Detailed Design and Visual(s)

Building upon the last section, we will further describe in detail the authentication flow, backend management of API requests, the tracking of user groups, and certain frontend features.

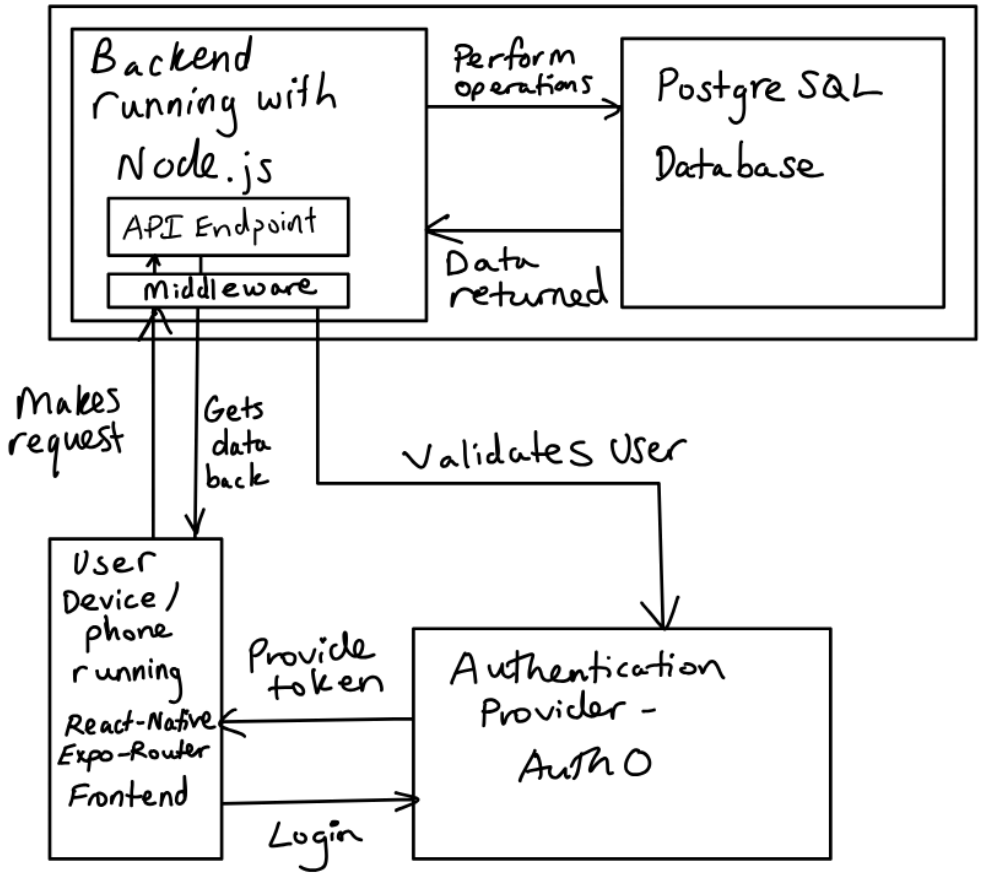
The authentication flow is as follows. First, the user clicks the login button, which takes them to a login page hosted by auth0. Auth0 will then manage the authentication of the user, regardless of whether they choose standard login (with credentials) or a third-party sign in. Auth0 requires unique configuration for use with iOS and android apps. Once the user returns from the hosted login, a specific auth0 state will be set with the user's credentials. Upon checking these credentials, the user will either be logged in and redirected to the app's pages or will be prompted to log in again. If the user is successful, then auth0 will provide a token and other values specific to the user. Now, on any API requests the frontend makes to the backend, it will include the auth0 token in the request header.

Following an API request from the frontend, the backend utilizes middleware to progress the user's request. The middleware serves as a middleman between API endpoints and frontend requests. Its job is to authenticate requests made to endpoints. It does this with the help of Express, which allows you to dictate functions that should be run before endpoints are entered. The app utilizes Express to attach the middleware to each and every API endpoint. Our middleware checks that the token and other details provided in the request header are what Auth0 expects. If auth0 verifies the provided tokens, then a request may proceed to the desired endpoint.

Another feature that our app provides is the ability for bar owners to manage details specific to their bar through a portal. This elicits a means to distinguish between a "normal" user and an "admin" user. This is accomplished through a roles table in our database. This could alternatively be accomplished through auth0, but such is a paid feature, so we have opted to implement it manually. A page for admin users to update details is also needed, so a page on the front end will be implemented that is only available to admin users.

Two notable front end features are maps and the admin portal. Maps is accomplished by using a maps library for expo. This library wraps another map API but makes it easier to integrate with our expo app. The admin portal is another page that will be implemented, with the only notable difference being an additional check with the middleware that verifies the user is of admin privileges.

Below is a diagram detailing the entirety of the system. It shows the general interactions that sub-systems and components of the system participate in.



4.3.3 Functionality

Ames After Dark functions as a cross-platform mobile application connecting users to Ames nightlife through real-time event discovery, social features, and business engagement. The system integrates a mobile frontend, backend API, database, and admin portal to deliver live, interactive updates for both users and bar owners.

When users open the app, they're greeted by the Home Page, a home feed displaying current events, drink specials, and featured venues. Widgets show bar names, event titles, specials, and friends. Users can tap any of these to be directed to the corresponding page for more details.

The Map Page provides a real-time map of Ames, with pins for bars and mutual friends' locations (if the user has location sharing enabled). Users can navigate directly to a bar, apply filters to show specific venues, and visualize bar crawl routes.

In the Gallery Page, users browse professional photos from nightlife events, filtered by date or venue. Photos can be viewed in full or downloaded by the user.

The Bars Page serves as a directory for all local venues, displaying details like operating hours, capacity levels, live deals, menus, and booking options. Users can also favorite bars to pin them to the top of their list.

Through the Friends Page, users can manage and view their friends. Being "friends" with a person means that they can see the user's location, as long as the user has opted in to the location sharing feature. You can also "check-in" with a friend, being notified when your friend arrives home.

Through the Account Page, users manage profiles and app settings, including notification and location preferences.

The Admin Portal lets bar owners securely update specials, events, bar details, and photos. All changes are synced directly to the app, keeping data accurate and timely.

Overall, Ames After Dark acts as a nightlife companion application, helping users discover events, stay connected, and support local businesses in one mobile experience.

4.3.4 Areas of Concern and Development

Based on our current design, we feel our system will satisfy the key requirements and user needs. The Node.js and Express backend provides a strong foundation for scalability, real-time data handling, and integration with mobile and web clients. Our architecture supports essential features such as user authentication, event listings, and location-based recommendations, which align closely with the expectations of users looking for nightlife options nearby.

However, there are still some concerns and challenges that we have identified moving forward:

- **Performance & Load Handling:** As the app grows, handling simultaneous user requests and maintaining quick response times could become challenging, especially with our current server constraints.
- **Data Accuracy & Reliability**: Ensuring that event and venue data remains accurate and up to date will be important for maintaining user trust.
- **Security & Authentication:** Ensuring that our app stays fully authenticated is an ongoing challenge with the priority of security and user trust.
- **User Experience Integration**: Making sure that the frontend and backend communicate smoothly to deliver a fast, intuitive experience remains a key development focus.

Our immediate plans to address these concerns include:

- Setting up automated testing to ensure reliability and prevent regression.
- Implementing database indexing and caching to improve performance under load.
- Integrating background security checks within the app.

We also have a few questions for our clients, TAs, and faculty advisers as we continue development:

- Are there specific privacy or data retention requirements we should follow for user information?
- What metrics or features should we prioritize to best measure user engagement and satisfaction?
- Would it be beneficial to include real-time features at this stage, or should we focus on core functionality first?
- How much emphasis should we place on scalability and long-term deployment versus short-term usability for this iteration?

Overall, while our design meets the initial requirements well, our next steps focus on refining performance, improving data reliability, and validating our design through user and client feedback to ensure that the final product truly meets expectations.

4.4 Technology Considerations

Ames After Dark uses technology appropriate to its developers, with the goal of maximizing development speed, minimizing learning curves, and prioritizing simple technologies over complex ones.

Frontend: React Native with Expo

React Native enables cross-platform mobile development for both iOS and Android from a single codebase, reducing development time and ensuring consistent performance across devices. Expo simplifies testing, debugging, and deployment.

- **Strengths:** Only necessary to write one codebase for both devices
- **Weakness:** Occasional performance limitations for highly complex animations or device-native features.
- **Trade-Off:** Chosen for faster iteration and broader accessibility over full native performance.

Backend: Node.js with Express

The backend server is built with Node.js using the Express framework, which handles API requests, authentication, and data synchronization. Its event-driven, non-blocking architecture supports multiple users efficiently.

- **Strengths:** High scalability and ease of implementing middleware
- **Weaknesses:** Single-threaded nature may limit CPU intensive operations.
- **Trade-Off:** Chosen for lightweight, fast API handling rather than computational processing.

Database: PostgreSQL

PostgreSQL serves as the primary database system, managing structured data for users, bars, events, and photos. Its relational schema ensures data integrity through foreign keys, indexing, and great query capabilities.

- **Strengths:** High data reliability, ACID compliance, strong support for complex relationships.
- **Weaknesses:** Slightly higher configuration complexity than NoSQL alternatives.
- **Trade-Off:** Chosen for relational stability and security over flexibility in unstructured storage.

Server & Management Tools: Apache, Cockpit, PM2, PG Admin

The backend is hosted on an Apache server with Cockpit providing GUI-based web management for the environment. PM2 is used for process monitoring and maintaining server uptime, while PG Admin facilitates database administration and query testing.

- **Strengths:** Reliable deployment stack with clear process monitoring and administrative control.
- **Weaknesses:** Slightly more setup overhead compared to integrated hosting platforms.
- **Trade-Off:** Chosen for better system transparency and administrative flexibility.

Mapping & Geolocation: Expo Location and React Native Maps

Expo Location and React Native Maps provide precise geolocation services for mapping bars and friends' check-ins, offering routing, clustering, and visual customization.

- **Strengths:** Very easy to use, well-documented, integrated with Expo tasks
- **Weaknesses:** Requires using the Expo framework, less configurable than the dedicated library

- **Trade-Off:** Chosen for the easy integration with Expo and flexibility when adding new libraries

Overall, the chosen technologies ensure that Ames After Dark development time is spent writing code rather than getting frustrated with technologies that are meant for more complex applications. The design reflects careful consideration of technical trade-offs, user needs, and ease of development.

4.5 Design Analysis

So far, we have made substantial progress in both the frontend and backend development of our nightlife app. On the backend, we have implemented a Node.js and Express server that connects to our database and supports several core API routes, including user registration, login, and event retrieval. We have successfully integrated JSON Web Token (JWT) authentication to secure user sessions and verified that protected routes correctly restrict access to authorized users.

On the frontend, built with React Native, we have implemented several main screens that form the foundation of the user experience:

- **Home Page:** Displays featured events and recommendations based on user preferences.
- **Photo Gallery Page:** Allows users to view and download photos from different bars in Ames, Iowa.
- **Maps Page:** Integrates with react-native-maps to show nearby bars, clubs, and events based on the user's location.
- **Account Page:** Lets users manage their profile details and app settings.
- **Friends Page:** Displays a user's connections ("friends") and enables viewing their locations and allows check-ins.

We have successfully tested navigation between these pages and verified communication between the frontend and backend for login and event data retrieval.

Our proposed design from Section 4.3 has worked effectively so far. The RESTful API structure allowed smooth integration with the React Native frontend, and the modular Express backend has made development and debugging straightforward.

However, several challenges and areas for improvement have emerged:

- **API Response Speed:** Some routes, especially those fetching multiple event records, can be slow and may need database optimization or caching.
- **Data Synchronization:** Keeping map and gallery data updated in real time across devices is still being refined.
- **UI Consistency:** We are still finalizing a consistent style and layout across pages to ensure a cohesive user experience.
- **Testing:** We have no automated testing so far, and very little structure surrounding the manual testing process.

To address these issues, our next steps include:

- Optimizing database queries and implementing caching for faster data retrieval.
- Refining the UI and navigation flow for a smoother and simpler user experience.
- Adding unit and integration tests for both backend endpoints and frontend components.

Overall, our progress so far validates the feasibility of our design. The chosen technologies work well together, and our architecture supports scalability and flexibility for future features. The remaining issues primarily involve performance tuning and feature refinement, not fundamental design flaws.

5 TESTING

5.1 Unit Testing

Unit testing for Ames After Dark focuses on verifying that individual components, backend endpoints, and database operations behave correctly in isolation. Mock data simulated API inputs, and controlled test environments will be used to validate expected behavior before full system integration.

Frontend Testing (React Native App)

Frontend unit tests ensure that UI components render correctly, update state appropriately, and behave predictably with mock data replacing backend responses. Planned unit tests include:

Tonight Page Components

- Verify that the Open Now filter displays only bars marked as “open” in mock data.
- Ensure event cards populate correct information (bar name, event, specials) from mock data.
- Confirm the Deals Tonight filter correctly displays bars with specials in the mock dataset.
- Validate that selecting Friends Near Me loads friends’ mock locations and displays them correctly (bar names).
- Check that tapping a bar card navigates to the correct Bar Profile page.

Map Components

- Confirm all mock bar locations appear as pins with correct data.
- Ensure tapping a bar pin opens the correct bar detail page.
- Validate bar crawl routes update correctly when mock route sequences change.

Gallery Components

- Verify that gallery thumbnails render correctly using mock SmugMug API responses.
- Confirm photo modals load the correct high-resolution image URL returned from SmugMug.
- Validate that saving a photo triggers the correct device permission request and handles denial/approval correctly.

Bar Profile Components

- Ensure menus, specials, and capacity indicators render correctly using mock bar data.

- Validate that the Map component routes the user to the Map Page and that the Gallery component routes to the bar's most recent album or the main gallery.

Account Page Components

- Validate input handling for profile edits (name, email, phone).
- Confirm toggles for notifications and location sharing update local state correctly.
- Ensure mutual friend logic shows correct mock friend connections.

Backend API Testing (Node.js Server)

Backend unit tests focus on route behavior, input validation, and business logic.

- **User Authentication**: Validate login logic, error handling for invalid credentials, and correct responses when accounts exist or do not exist.
- **Events and Deals Endpoints**: Confirm that API routes return the correct structured data for nightly events when provided with known inputs.
- **Bar Lookup & Filtering**: Ensure that filtering parameters (e.g., "open now") return the correct subset of mock data.

Database-Level Testing (PostgreSQL)

Database unit tests use test schemas or seeded data to validate relational logic.

- Validating foreign key relationships between bars, events, and photos.
- Ensuring inserts, updates, and deletes behave correctly for bar specials and uploaded photos.
- Testing friendship relationships through join tables to confirm that only mutual connections produce visible friend data.

Overall, unit testing will help ensure that each part of Ames After Dark functions independently and predictably before integration work begins.

5.2 Interface Testing

Interface testing verifies that different parts of the Ames After Dark system communicate correctly with each other. This includes interactions between the mobile app, backend server, PostgreSQL database, device services, and the SmugMug API.

Frontend & Backend API

Interface test will confirm that the mobile app correctly communicates with the server using structured API requests and responses. These tests check:

- Whether the frontend can successfully request nightly events, bar specials, menus, and photos from the backend.
- That invalid or incomplete requests return appropriate error messages rather than causing crashes.

- Correct handling of slow or failed network calls (e.g., fallback messages on the Tonight Page or Gallery Page).
- That the front end correctly processes location data returned by the API for displaying friends and bar pins on the Map Page.

Backend API & PostgreSQL Database

These tests ensure that the backend retrieves and updates data stored in PostgreSQL correctly. Planned tests include:

- Verifying that events, bar details, and photos stored in the database are returned in the correct format expected by the frontend.
- Confirming that new specials, deals, and photos in the admin portal are properly stored and immediately retrievable.
- Checking that invalid SQL operations (e.g., inserting a booking with missing fields) produce clean, safe errors.

Admin Portal & Backend API

Because bar owners update much of the app's content, testing this interface is critical. Test will ensure:

- Admin uploads (specials, events, photos) correctly trigger API calls and save them to the database.
- Bar profile edits propagate to the mobile app without inconsistencies.
- Error states appear correctly in the admin interface when uploads fail.

Frontend & SmugMug API (Gallery Page)

The Gallery Page relies on external image hosting, so integration must be validated.

- Confirm correctly formatted requests to the SmugMug API return expected datasets.
- Validate that gallery thumbnails match photo data provided by SmugMug.
- Ensure that pagination or date-filtered requests return proper filtered results.
- Verify that full-resolution URLs provided by SmugMug load properly in the viewer.

Frontend & Device Services (Maps, Geolocation, Permissions)

Interface testing will also validate how the app interacts with device-native tools. This includes:

- Validate permission request handling and logic for showing/hiding friend locations.
- Confirm map rendering, pin placement, and routing coordinate transfers to navigation apps.
- Ensure the app requests the correct OS-level permissions when saving photos locally. Confirm behavior when permissions are denied.

Together, these interface tests ensure that Ames After Dark operates as a cohesive, reliable system across all major components and user pathways.

5.3 Integration Testing

Integration testing for Ames After Dark will focus on the end-to-end paths that cross the React Native frontend, the Node.js/Express backend, the PostgreSQL database, authentication, and external services such as our photo hosting and mapping/location tools. These tests build directly on the unit and interface tests described in Sections 5.1 and 5.2 and are intended to confirm that data and control flow correctly across the whole stack

A first critical path is user authentication and protected data access. When a user logs in through our authentication flow (Auth0), the frontend will receive tokens that are attached to subsequent API requests. Integration tests here will cover logging in, calling protected backend routes with valid and invalid tokens, and confirming that the backend middleware correctly allows or rejects requests before they reach the core endpoints. This verifies that the authentication, backend, and database layers behave consistently for real users.

A second critical path is the Tonight Page loading data from the backend and database. The app will request nightly events, bar specials, and status information from the backend, which in turn queries PostgreSQL. Integration testing will exercise this full pipeline by starting from the Tonight Page in the app or simulator, triggering real API calls, and verifying that the data returned from the database is rendered correctly in the feed and filters. This directly supports the requirement for a real-time events and specials feed.

A third critical path is the Map Page combining device location, bar data, and friend/location information. The frontend uses Expo Location and React Native Maps, while the backend and database supply bar and friend location data. Integration tests will run on iOS/Android simulators and test devices, enabling location services, loading map data from the backend, and confirming that bar pins and friend locations appear and update as expected. This validates that device permissions, geolocation, the backend, and the database all work together.

A fourth critical path is the Gallery Page retrieving photos through the backend and external photo hosting, as described in earlier sections. The app will request photo metadata from the backend, which will in turn call out to SmugMug our chosen photo provider. Integration tests will verify that the Gallery Page can load lists of

photos, open individual images, and respect filters by bar and date based on the combined behavior of the backend, database, and external service.

Finally, a key integration path involves the admin portal updating data that is then consumed by the mobile app. Bar owners will use the portal to create or modify specials, events, photos, and bar profile details. Integration testing will cover making these changes through the portal, confirming that they are stored correctly in PostgreSQL via the backend, and then verifying that the Tonight, Bars, and Gallery pages in the app display the updated information after a refresh. This closes the loop between bar owners' workflows and the end-user experience.

These integration tests will be layered on top of our Jest-based unit tests and the interface testing we perform with real API calls. They will be run on simulators, test devices, and through our CI/CD setup so that critical end-to-end flows are exercised whenever changes are pushed.

5.4 System Testing

System-level testing will verify that the frontend, backend, middleware, and cloud services are functioning correctly as one system. Unit testing will use Jest on both frontend and backend to validate core logic such as authentication, retrieving bar data, rendering UI components, and handling friend requests. Interface testing will focus on ensuring that the app communicates correctly with backend APIs, using tools like Insomnia (Postman alternative) and automated scripts to confirm proper request formats and error handling. Integration testing will then evaluate complete end-to-end scenarios such as logging in and viewing a bar's page and viewing gallery images from a specific bar and day, or adding friends and seeing updates reflected in the interface. Tools such as Detox may be used for automated mobile interaction tests, with all tests run through our GitHub Actions CI/CD pipeline to maintain consistency and reliability. This strategy ensures that the system meets all functional and non-functional requirements at the full application level.

5.5 Regression Testing

Our main method of checking that new additions do not break existing functionality is by using CICD. GitHub allows us convenient functionality for ensuring that all new code adheres to the standards that have already been set. By using GitHub actions, we can have specific tests run on each push to a dedicated branch using our runner. That way we can be reassured that when it's merged into main and deployed it does not break anything.

While there are no dedicated regression tests, our tests in the aggregate serve as regression tests when run on branch pushes. Our plan for bugs is to create a test that fails on the old branch but passes on the new branch. This way we can be sure that the test accurately covers the bug we were attempting to solve. This also prevents regressions in the area on future changes because failures would be caught by CICD.

Our critical features include core functionality such as auth, bar information, friends, map, gallery, etc. This core functionality is driven largely by requirements, although tools play a role in it as well. Features with tools that make testing easier and more convenient will be prioritized. Finding and fixing bugs also contributes heavily to the direction that we take in developing regression tests. Things more likely to be broken by future changes should be prioritized.

5.6 Acceptance Testing

Acceptance testing for the application will verify that the app's functional and non-functional requirements are met through scenario-based tests, performance checks, and client review. Functional requirements such as the friends feature or bar hours and deals display will be tested using scripted scenarios to ensure each feature behaves exactly as specified on both Android and iOS devices.

Non-functional requirements, including performance, usability, and reliability, will be demonstrated through testing on the backend server for speed, device compatibility testing and emulation for the React Native frontend, basic testing with failed authentication, and usability walkthroughs with representative users.

Our client is a developer on our team, allowing continuous involvement throughout the project. This means they are hands-on each step of the way, and feedback is given incrementally. The team incorporates the client's feedback during each iteration, at the end of the year there will be a final sign-off from our client which will ensure the system meets the client's expectations and real-world needs.

5.7 Security Testing

Security testing will become a small but important part of our test suite. Since we have chosen to implement auth using a third party (auth0), the scope of security testing we can do is somewhat limited.

Something that we will test is that unauthenticated users cannot access any of the app's endpoints or content. This is simple to test and must only verify that our middleware stops unauthenticated requests. Another thing to test is that users

cannot navigate to specific screens without being authenticated, and that if they do navigate, no information is displayed, and they are navigated back to the login page.

Another important part of security is making sure that no secret key or other API key is written in code in the repository. Simple but powerful measures can be taken to prevent the use of secrets, like searching for certain patterns throughout the repo in our CICD pipeline.

5.8 User Testing

Our team will be implementing live-session beta testing with a select group of pre-selected users. We will be rolling out the app onto TestFlight for iOS and potentially deploying an SDK file for Android users. This will give our identified users a chance to use the app in a production setting and test all our features in their own time; as opposed to lab-style environments.

Users will be able to have the app on their own devices and go out for the night, getting optimal use out of the directories, map, and other key features. Every user will have the ability to give us detailed feedback on their personal experience, bugs they wish to report, and design improvements we can make as a team. This will be our final stage before deploying to the public, and one of the most time and resource intensive parts of our project; requiring active system monitoring and incident response responsibility.

5.9 Results

Our tests thus far have been limited. In total we have eight tests, all of which are on the backend and succeed. The tests on the backend include several tests for both the controllers and services. Some of our testing for UI related software is qualitatively based and ensures that the screens will be aesthetic and intuitive to users.

Our backend tests demonstrate compliance with requirements by ensuring that they function according to expectations and will perform their role throughout the application. Our limited intuitive/aesthetics testing makes sure that users are engaged with our application and satisfied with their experience.

We have learned that tests in a mobile app setting can be slightly challenging. While backend tests are simple and relatively easy to implement, UI tests require more sophisticated setups and tools if you want to cover more than just components or similar things in the frontend. Next steps for testing include expanding test coverage for the backend and improving testing ability for the frontend.

REFERENCES

[1] J. S. Zhang, M. Gatrell, R. Han, Q. Lv, and S. Mishra, "GEVR: An Event Venue Recommendation System for Groups of Mobile Users," *arXiv preprint arXiv:1903.10512*, Mar. 2019. [Online]. Available: https://arxiv.org/abs/1903.10512. [Accessed: Oct. 20, 2025].

[2] S. Zhang, K. Alanezi, M. Gartrell, R. Han, and Q. Lv, "Understanding Group Event Scheduling via the OutWithFriendz Mobile Application," *arXiv preprint arXiv:1710.02609*, Oct. 2017. [Online]. Available: https://arxiv.org/abs/1710.02609. [Accessed: Oct. 20, 2025].

[3] N. Singh, "A Critical Examination of Eventbrite Reviews and Insights," ReviewNext, 2024. [Online]. Available: https://reviewnex.com/articles/eventbrite-reviews-analysis/. [Accessed: Oct. 20, 2025].4.1.3 Technical Complexity