Campus Resource Hub PRD

CONTEXT

Campus Resource Hub is a full-stack web application that enables university departments, student organizations, and individuals to list, share, and reserve campus resources (study rooms, lab equipment, AV equipment, event spaces, tutoring services).

Technical Stack: Flask (Python 3.10+), Jinja2 templates, Bootstrap 5, SQLite/ PostgreSQL, deployed on AWS Elastic Beanstalk. Demonstrates AI-first development using Cursor and GitHub Copilot.

PROBLEM STATEMENT

Universities struggle with fragmented resource management systems using spreadsheets, email chains, and physical sign-up sheets, leading to resource underutilization, booking conflicts, poor discoverability, and excessive administrative burden. There is no centralized platform for students to find and book resources, or for staff to efficiently manage their assets. This results in wasted resources, frustrated users, and limited accountability across campus.

GOALS & NON-GOALS

GOALS: Centralized resource management (study rooms, lab equipment, AV, event spaces, tutoring, other), efficient booking system with calendar integration and conflict detection, role-based access control (Student, Staff, Admin), user engagement via ratings/reviews and messaging, administrative tools with analytics dashboard and AI-powered query assistant, data-driven insights with aggregate ratings and usage stats.

NON-GOALS: Payment processing, mobile native apps (web-only), real-time features, advanced scheduling (waitlists, complex recurring patterns), external system integrations (SIS, calendar sync, SSO), resource maintenance tracking, multi-tenancy, advanced analytics (predictive ML).

USER STORIES

As a student, I want to browse and search for available resources so that I can find study spaces, equipment, or services I need.

As a student, I want to book resources with date/time selection and conflict detection so that I can reserve them without double-booking myself.

As a student, I want to write reviews after a booking so that I can provide feedback and help other users.

As a student, I want to message resource owners and receive booking notifications so that I can communicate about reservations and stay informed.

As a staff member, I want to create and manage resource listings so that students can discover and book my resources.

As a staff member, I want to approve or reject booking requests so that I can control access and prevent conflicts.

As an administrator, I want to manage users, resources, and bookings so that I can oversee platform operations.

As an administrator, I want to moderate reviews and view analytics so that I can maintain content quality and understand usage.

As an administrator, I want to use an AI chatbot for natural language analytics queries so that I can quickly get insights without writing SQL.

USER FLOWS

1. Student Books Resource: Login → Browse → View Details → Select Date/Time → Conflict Check → Submit → Receive Notification → View in My Bookings

2. Staff Approves Booking: Login → My Resources → View Pending Requests → Approve/Reject → Requester Notified

3. Admin Uses AI Analytics: Login → Admin Panel → Open Chatbot → Ask Question → View AI-Generated Answer with SQL Query

SUCCESS METRICS

User Adoption: 100+ registered users (month 1), 20% DAU, 60% MAU, 40% 7-day retention.

Engagement: 20+ resources created, 50+ bookings/month, 30% review completion rate, 2 messages per booking average.

User Satisfaction: 4.0+ average rating, 80%+ approval rate, <10% cancellation rate, <5% support ticket rate.

Administrative: <24hr average approval time, weekly admin dashboard usage, 10+ AI chatbot queries/week.

RISKS

Technical: Database performance (SQLite scaling) → PostgreSQL migration ready; Security vulnerabilities → ORM, validation, hashing, audits; API rate limiting (Gemini) → Retry logic, fallback queries; File upload security → Type/size validation, S3 for production; Deployment issues → Documentation, staging, rollback procedures.

Product: Low user adoption → User-friendly design, training, gradual rollout; Resource owner resistance → Simple process, clear benefits, admin support; Data quality → Required fields, audits, moderation; Booking conflicts → Robust algorithm, warnings, admin override.

Business/Operational: Maintenance burden → Documentation, automated testing; Cost overruns → Free tier, monitoring, optimization; Compliance (FERPA, GDPR) → Secure storage, encryption, privacy policy, audits.

External: Third-party failures (Gemini, AWS) → Graceful degradation, multi-AZ, monitoring; Technology obsolescence → Regular updates, migration planning.