CS673 Software Engineering

Team 4: Kickaas

Project Proposal and Planning

| **Team Member:** | **Role(s):** | **Signature:** | **Date:** |
| --- | --- | --- | --- |
| Javier | Team Leader | *Javi* | 09/19/2025 |
| Vamsi | Requirement Leader | *Vamsi* | 09/23/2025 |
| Johannes | Design and Implementation Leader | *Johannes* | 09/20/2025 |
| Saniya | Configuration Leader | *Saniya* | 09/20/2025 |
| Yuanfei (Keni) | QA Leader | *Keni* | 09/23/2025 |
| Shreya | Security Leader | *Shreya* | 09/20/2025 |

**Revision History**

| **Version:** | **Author:** | **Date:** | **Change:** |
| --- | --- | --- | --- |
| 0 | Team | 09/19/2025 | Initial Draft |
| 1 | Team | 10/09/2015 | Feedback From Iteration 0 |
|  |  |  |  |
|  |  |  |  |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High Level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management](#_tk7yixobah8p)

[Timeline](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_odzub6q8z5xd)

[Code Commit Guideline and Git Branching Strategy](#_9q1tgcflcd3j)

[Code Review Process](#_hx3eaiwb8v3m)

[CI/CD Plan](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[AI Usage Log](#_161vmjht87fy)

[Glossary](#_ty3i2nqffhtc)

# **Overview**

Modern social event discovery and hosting tools span a wide range from casual invite pages to full ticketing stacks. Many existing tools are either too heavy (enterprise ticketing) or too casual (single-use invite pages), and they often trade off privacy, discoverability, and simplicity. Our project aims to fill a middle ground: a lightweight, privacy-conscious event hosting and discovery website that makes it quick to create attractive event pages, manage RSVPs, and integrate with calendars — while remaining easy to extend. The purpose is to let organizers create event pages and guest lists quickly, let guests RSVP and share the event, and provide organizers lightweight analytics and collaboration tools for running events.

**Potential Users**

* Students and campus communities organizing parties, club meetings, and socials.
* Small local organizers (house concerts, pop-ups, meetups).
* Friends planning private gatherings (birthdays, dinners).
* Community groups or small non-profits that want an easy invite-and-RSVP flow without heavy fees or ads.
* Developers/teams who want an open, self-hostable alternative for internal events.

**Basic Functionality (Minimal Viable Product)**

* Create an event with title, time, description, image …
* Visualize and scroll through all events
* Visualize a single event

The team aims to have a really simple and minimal MVP up and running in order to avoid scope creep and complexity. The team understands the initial MVP is really simple and barely provides any functionality. However, the team believes this is the right approach for this type of project where new features can easily be added based on user and developer feedback.

**Technology Stack**

For the teck stack the team decided to go with 2 simple frameworks both in the frontend and backend.

1. Frontend:
   1. Next.js 15 (App Router), React 19, Tailwind v4
2. Backend:
   1. Python 3.11 (FastAPI) + Uvicorn for REST API
3. Storage:
   1. PostgreSQL (relational model for events, attendees, RSVPs) for local development and production database.
   2. SQLite for testing.
4. Auth: OAuth2 / OpenID Connect (e.g., Auth0, Keycloak) or JWT for account sessions.
5. CD:
   1. Docker images
   2. Docker containers
6. CI
   1. Github Actions
   2. Ruff + Tox
   3. Uv
   4. ESLint + Prettier with accessibility plugin
7. Observability: Prometheus + Grafana for metrics and logs.

# **Related Work**

We examined existing online services to understand features, user expectations, and gaps we can address.

**Partiful** — modern, casual event pages with emphasis on quick invites and text-based sharing. Partiful focuses on visually appealing invite pages, text blasts/auto-reminders, and a simple host/guest flow aimed at informal social gatherings. It offers settings like password-protected events, anonymized guest lists, and payment collection via Venmo/PayPal for cost sharing.

**Eventbrite** — a full-featured ticketing and event-management platform for paid and free events. Eventbrite emphasizes ticket sales, seat assignment, payment processing, advanced reporting and organizer tools; it’s geared toward professional events and large-scale ticketing with fees and deep integrations. If your use case requires robust ticketing, seat maps, or enterprise reporting, Eventbrite is a natural choice; however, it’s heavier and fee-focused than what we intend for casual or self-hosted use.

**Meetup** — community-oriented platform for recurring groups and meetups. Meetup’s strength is group discovery, recurring scheduling, and community-building features (cross-group messaging, member management). For communities that need group infrastructure and discovery across topics, Meetup is powerful but more opinionated toward group organization rather than single-event simplicity.

**Facebook Events** — built into a large social graph; great for discovery and promotion to an existing social audience. Facebook’s advantage is reach and event promotion via Pages and ads, but it trades off control and privacy; it also requires users to be on the Facebook platform.

**Evite** — classic online invitations with many themed templates and RSVP tracking, oriented to casual private events and family gatherings. Evite offers premium templates/features, read receipts, and a straightforward invitation flow. It’s template-heavy (good for formal invites) but less modern in social discovery and mobile-first share flows.

**Splash** — an event marketing and management platform focused on professional event marketers (virtual/hybrid/in-person) with analytics and brandable pages. It’s feature-rich for marketing and data-driven event teams, but overkill for small social events and not optimized for lightweight self-hosting.

How our project differs / the niche we target:

1. **Lightweight, modern UX for casual events:** we’ll adopt the quick-create, mobile-first flow popularized by Partiful, but add configurable options for slightly more structured events (CSV export, simple analytics) without the complexity or fees of Eventbrite.
2. **Open and extensible stack:** built on standard web primitives (REST API + SQL) so other developers can extend integrations (calendar sync, campus SSO, custom notifications) — unlike closed hosted platforms.
3. **Focused scope:** prioritize speed of creation, shareability (link/SMS), rather than full enterprise ticketing or large-marketing features (which Eventbrite and Splash focus on).
4. **Affordable/basic payments integration for cost-sharing:** provide lightweight payment links for chip-in scenarios (Venmo/PayPal) but not full ticketing/payment processing out of the box (keeps complexity down). Partiful similarly supports chip-in links; we aim to match this behavior while keeping open-source hosting options.

# 

# **Proposed High Level Requirements**

**Functional Requirements (User Stories)**

**Essential Features:**

1. Unified Role Model
   1. Any logged-in user can create/manage events (organizer actions) and register to events (attendee actions).
   2. Not a separate profile. No separate “organizer account” needed.
   3. Permissions:

* A user may edit/delete only events they created.
* A user may register to any public event (Edge case:- optionally to their own event).
* Private events require an invite.

1. Auth and Profile
   1. As a user, I want to sign up/log in, so that I can create events and register to others’ events from one account.
2. Organizer Actions
   1. As a user I want to create an event (title, time, description, image, location), so that I can host it.
   2. As a user, I want to update or cancel my event, so that I can fix details or close it if needed.
   3. As a user, I want to update the event my location will take place, so that I can get more visibility.
3. Attendee Actions (same user)
   1. As a user, I want to browse/search/filter/sort public events, so that I can find ones I care about.
   2. As a user, I want to open an event page with details, so that I can decide to attend.
   3. As a user, I want to register/RSVP for a public event and see my registrations, so that I can manage my plans.
   4. As a user, I want to see the public events on a map, so that I can decide what event to attend based on proximity.
4. Digital Ticket Delivery
   1. As a user, I want to receive my invite via email or text-messaging.

**Desirable Features:**

1. Event Analytics
   1. As a user, I want lightweight analytics (views, RSVPs, …)
2. Payment Processing (Will use Stripe/ PayPal etc.)
   1. As a user I want to pay securely via an online gateway, so that my ticket purchase is confirmed.

**Optional Features:**

1. Private Events
   1. As a user, I want to mark events private and invite specific people, so that only invited users can view and assist the event.
2. Assigned Seats
   1. As a user, I want to be able to have assigned seating for more formal private events.

**Non-Functional Requirements**

**Essential Non-Functional Requirements**

Performance (MVP Critical)

* Page loads within 5 seconds (relaxed for MVP)
* Event search results within 3 seconds
* Support 100 concurrent users minimum

Security (Must-Have)

* HTTPS/TLS encryption for all data transmission
* Clerk authentication integration
* Basic session management and timeouts
* Input validation and SQL injection prevention

Usability (Core UX)

* Mobile-responsive design (essential for event apps)
* Browser support for Chrome, Firefox, Safari latest versions
* Basic accessibility (keyboard navigation, alt text)
* Intuitive navigation structure

Reliability (Baseline)

* Basic error handling and user feedback

**Desirable Non-Functional Requirements**

Enhanced Performance

* Page loads within 3 seconds
* Support 500 concurrent users
* Database query optimization
* Advanced analytics and reporting

Enhanced Security

* Multi-factor authentication
* Advanced rate limiting
* Data encryption at rest
* Comprehensive audit logging

Improved Usability

* Progressive Web App features
* Advanced filtering and search capabilities
* Real-time notifications

Monitoring & Observability

* Prometheus metrics integration
* Structured logging
* Performance monitoring dashboards

**Optional Non-Functional Requirements**

Premium Performance

* Sub-2 second page loads
* Support 1000+ concurrent users
* Auto-scaling capabilities
* Global CDN integration

Advanced Security

* Advanced threat detection
* Penetration testing
* Security headers optimization

Premium Features

* Offline capability
* A/B testing framework
* Multi-language support

# 

# **Management Plan**

## Objectives and Priorities

1. Deliver all essential features for a functional MVP
   1. Event creation and management (title, date, venue, capacity)
   2. Basic attendee registration

These core features must be completed first, as they are the foundation of all other work

1. Ensure successful deployment and accessibility
   1. Deploy the application early to target hosting platform
   2. Dockerize Backend, frontend and database and verify connectivity

Guarantees we identify infrastructure issues before later iterations

1. Add desirable features to improve usability
   1. Online ticket purchase with Stripe and PayPal, digital ticket delivery, organized dashboard and multi-ticket bundles

These will be built in Iterations 1 and 2 once the MVP is stable

1. Maintain high software quality
   1. Enforce code reviews, linting and automated testing (unit, integration, end-to-end)

Track and resolve critical bugs before feature enhancement

1. Optimize user experience and security
   1. Ensure smooth performance under load
   2. Protect user data with secure auth, database constraints
2. Implement optional enhancements if time allows
   1. Analytics (revenue by ticket type, attendance forecasting, Organizer analytics)
   2. Polished UX improvements beyond the MVP

## 

## Risk Management

The main issues identified for our project include issues with third-party integrations (such as Stripe payments, email ticket delivery and map services), potential data integration problems (like overbooking), performance issues, and security or privacy concerns involving attendee data. Additional risks involve environmental drift between local and production deployments (we try to mitigate this as much as possible with dockerized environments). To manage these risks, we prioritize early testing of critical integrations (payments, email, authentication), enforce database constraints, implement CI/CD pipelines with automated tests (both made by us and using Github’s copilot) and use proven third-party providers with fallback options. We will maintain a living risk register in a shared Google Sheet, update risks weekly or biweekly and ensure we create plans for high-impact issues.

**Risk Management Sheet Link:** [**CS673\_SPPP\_RiskManagement\_team4**](https://docs.google.com/spreadsheets/d/1h3qV7N6FovAr7cUPWUP1A71bieeB8etOuxzvd29D1MY/edit?gid=0#gid=0)

## Timeline

| **Iteration** | **Functional Requirements (Essential/Disable/Option)** | **Tasks (Cross Requirements Tasks)** | **Estimated/Real Total Person Hours** |
| --- | --- | --- | --- |
| 1 | Event Creation & Management  User authentication and Registration  Event registration | Repo setup & CI/CD pipeline  Register for public events using email  Authenticate users and store in DB (Google Auth)  Design database schema (users, events, tickets)  Deploy skeleton MVP | ~ 80-100 hrs |
| 2 | Ticketing & Online Purchase  Private Events (invites  Payment Processing (Stripe/Paypal)  Digital Ticket Delivery | Build sales tracking dashboard  Mock user payment and ticket selection flows  Integrate Stripe (sandbox mode)  Tokenized invite links for private event registration  Configure email ticket delivery  Unit + integration tests | ~ 120-150 hrs |
| 3 | Organizer Dashboard (Sales Overview)  Map based filtering  Multi-ticket Bundles  Advanced Analytics (if time allows) | Optimize UI/UX and performance  Final bug fixing + production deployment  Implement multi-ticket group purchase | ~ 100-120 hrs |

# **Configuration Management Plan**

## Tools

**Version Control**

* Git and GitHub: Primary version control system for managing code changes, tracking history, and collaborative development
* Branch Protection Rules: Configured on main branch to require pull request reviews and status checks
* Pull Request Templates: Standardized templates for consistent PR descriptions and review criteria

**Development Environment**

* VS Code: Primary IDE with integrated Git support and extension ecosystem
* Cursor: AI-powered IDE for enhanced development productivity
* Extensions: Standardized extension pack including ESLint, Prettier, TypeScript, and Python support

**CI/CD Pipelines**

Continuous Integration

* Frontend
  + ESLint: JavaScript/TypeScript static analysis for code quality enforcement
    - Configured with Next.js and TypeScript rules in eslint.config.mjs
    - Enforces code standards, catches potential bugs, and maintains consistency
  + Prettier: Code formatting tool with shared configuration
    - Ensures consistent code style across the team
    - Integrated with ESLint for seamless formatting
* Backend
  + Ruff: lint and format checks
  + Tox: unify running environment for ci tasks
  + Uv: manage python dependencies and virtual environments
* GitHub Actions: Automated CI pipeline for testing, linting, and building
  + Triggers on pull requests and pushes to main branch
  + Runs frontend and backend test suites
  + Runs frontend and backend linting and formatting checks
  + Performs security scans and dependency checks
  + Builds both the frontend and backend images

Continuous Deployment

* Docker: Containerization platform for consistent deployment environments
  + Dockerfile.frontend: Node.js Alpine container for Next.js application
  + Dockerfile.backend: Python 3.11 slim container for FastAPI application
  + Multi-stage builds for optimized production images
* Docker Compose: Containerization of PostgresSQL database using docker containers and volumes
  + Used for both local development and production environments.
* Container Registry: Docker Registry for storing and distributing images

**SAST/DAST:**

* **SAST:**
  + Semgrep and GitHub CodeQL as SAST tools. Semgrep and GitHub CodeQL are used for scanning the code for vulnerabilities prior to running the code.
* **DAST:**
  + OWASP ZAP as a DAST tool. OWASP Zap is used to test the running website by sending automated requests and checking for issues like XSS (cross site scripting), SQL Injection and broken authentication.
  + Npm audit and pip-audit to regularly scan project dependencies for known security vulnerabilities

**AI:**

* GitHub Copilot to help with any Pull Request in GitHub, and help with any specific changes required before the request is approved.
* ChatGPT to help with any knowledge gaps, coding syntax, and other coding or presentation related work.
* Cursor IDE and VSCode + Copilot to increase the developers productivity.

## 

## Code Commit Guideline and Git Branching Strategy

**Coding Standard**

We will follow simple coding standards to keep the code clear and consistent:

* Frontend: ESLint + Prettier for JavaScript/Next.js code.
* Backend: PEP8 style guide with Ruff for Python code.
* Use clear variable and function names.
* Avoid unnecessary comments by simplifying the code and making it as readable as possible.
* Add comments for complex logic.
* Team members must use the [conventional style](https://www.conventionalcommits.org/en/v1.0.0-beta.2/) for commit and pull requests title and description.
* All commits must include a short but descriptive message.
* All pull requests must include a detailed description of the changes and their purpose. Developers should explain the why and not the what in their descriptions.

## Code Review Process

* All code changes will be submitted through GitHub pull requests.
* Repo is set so that the Copilot agent reviews all pull requests to help the team with code consistency.
* At least one teammate must review and approve each pull request before merging.
* Reviewers will check:
  + If the code is easy to read and follows the coding standards.
  + If there are enough comments and documentation.
  + If the code runs correctly and passes tests.
  + Feedback should be constructive and help improve the code.

**Branch Strategy**

* main: Production-ready code, protected branch
* develop: Integration branch for feature testing (if needed)

## 

## CI/CD Plan

**Continuous Integration Workflow**

1. Code Quality Checks:
   1. ESLint analysis for frontend
   2. Prettier formatting validation for frontend
   3. Ruff lint and format analysis for backend
2. Security Scans:
   1. Semgrep static analysis
   2. GitHub CodeQL security scanning
   3. Dependency vulnerability checks (npm audit, pip-audit)
3. Testing:
   1. Frontend unit tests using Jest/Vitest
   2. Backend unit tests using pytest
   3. Integration tests for API endpoints
   4. End to end tests for API endpoints
4. Image Build and Verification:
   1. Frontend Next.js build process
   2. Backend FastAPI application startup validation

**Continuous Deployment Strategy**

* Manual approval required for production releases
* Automatic rollback capability on health check failures

**Quality Gates and Approval Process**

1. Individual Development: Developer completes feature in personal branch
2. Self-Testing: Developer performs local testing and validation
3. Peer Review: 1-2 team members conduct code review via pull request
4. Automated Validation: All CI pipeline checks must pass
5. Merge Approval: Team Leader or designated approver merges to main
6. Deployment: Configuration Leader or designated approver manages deployment process

# 

# **Quality Assurance Plan**

## Metrics

| **Metric Name** | **Description** |
| --- | --- |
| **Test Case Pass Rate** | Percentage of test cases that pass during each iteration.  **Target:**100% |
| **Defect Count** | Number of bugs reported and tracked in GitHub Issues.  **Target:** ≤ 5 unresolved defects per iteration |
| **Code Review Coverage** | Percentage of pull requests that have been reviewed by at least one teammate.  **Target:** 100% |
| **User Stories Completed** | Number of user stories finished compared to the planned backlog.  **Target:** 80% |
| **Test Coverage** | Percentage of project code covered by automated unit and integration tests.  **Target:** ≥ 90% |

## Testing

* Unit Testing:
  + Each developer writes unit tests for the code they write.
* Integration Testing:
  + Each developer writes integration tests for the code they write.
* End to End Testing for the Frontend and Backend
  + Each developer writes end to end tests once a full workflow is completed. By doing so we avoid needing manual testing or at least keeping it to a minimum.
* Pact Testing
  + The team is still trying to decide if pact testing will be necessary to help keep the REST API backwards compatible and avoid any breakages between the frontend and backend.
* Tools:
  + Pytest for backend
  + Jest/React Testing Library for frontend
* Manual Testing:
  + Main features such as event creation, RSVP, and login will also be tested manually.
* Frequency:
  + Tests will run on every pull request using GitHub Actions CI, and a full regression test will be done at the end of each iteration.

## Defect Management

* Tool: GitHub Issues will be used to track defects.
* Types of Defects: Functional bugs, UI/UX issues, performance problems, and security issues.
* Process:
  + Report the bug in GitHub Issues with steps to reproduce.
  + QA Leader assigns the bug to a developer.
  + The developer fixes the bug and creates a pull request linked to the issue.
  + QA Leader verifies the fix, and then the issue is closed as the pull request is merged.

# 

# **AI Usage Log**

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

| **Tools:** | **Who:** | **Tasks:** | **Helpful?:** | **Evaluation/Modification:** | **Links:** |
| --- | --- | --- | --- | --- | --- |
| ChatGPT | Saniya | Asked for SAST/DAST tools that are compatible with our tech stack. | It was very helpful because it gave me a list of tools for both SAST/DAST. | I took ChatGPT’s suggestions and did a bit of research on each tool. I did not modify any of the content. | <https://chatgpt.com/share/68eeb060-1f94-8006-a6e6-f20060a3d31d> |
| ChatGpt | Yuanfei He | Ask Chat gpt about testing part | It provided some ideas on how to improve testing | I reviewed ChatGPT’s suggestions on testing and found them useful. I did not make major modifications but will consider refining the metrics and applying them during future iterations. | <https://chatgpt.com/share/68ee6e9d-d360-8012-a300-655f18774775> |
| ChatGPT | Vamsi | Asked for ColumnNames and Naming convention | It was helpful because it gave me the right naming convention for sql | I already had names for the columns but when I reviewed chatGPT’s suggestion it allowed for betting understanding than my own so I decided to use that. | Can’t find the link |
| ChatGPT | Shreya | Asked ChatGpt for Analytics that can be implemented in the project | It provided great ideas | I have taken ChatGPT’s suggestion into account. I haven’t implemented it yet but I have looked into it and plan to use it in future updates. | <https://chatgpt.com/share/68d45bb5-9d00-8006-96ad-886c9c530b01> |
| ChatGPT | Javier | Asked for a starter structure for the overview and related work sections | It provided a great template and structure to follow | Took part of ChatGPTs recommendation and adjusted it to fit our needs | Can’t find the link |
| CursorIDE | Javier | Helped with the backend and overall repo setup | NA | I already had a repo template I used for other python projects. However, cursor helped me adjust it for the new repo which has a bit of different structure | Cursos does not provide links |
| Copilot | Javier | Redo Configuration Plan Section | Very helpful | Gave me a broad idea of what to include based on the overall doc and our project. However, I had to adapt the style and structure to fit our needs | Can’t find how to share an open link with copilot |

# 

# **Glossary**

* MVP → Minimum Viable Product
* PR / Pr / pr → Pull request
* CI → Continuous Integration
* CD → Continuous Deployment