EventOnClick Phase 3 Finalization

Project Overview

EventOnClick is a web-based platform for discovering and managing local events. The project's goal was to enable event organizers to publish events and allow the public to easily find and attend them. The solution was developed using the Scrum agile methodology, with a strong focus on iterative delivery, measurable requirements, and modern cloud-native technologies.

Technical Approach

Methodology:

We adopted Scrum for project management, organizing work into four 2-week sprints. Each sprint had clear goals, a prioritized product backlog, and defined Scrum roles: Product Owner, Scrum Master, and a cross-functional Development Team. This approach enabled rapid feedback, continuous improvement, and adaptability to changes.

Requirements:

Functional and non-functional requirements were specified with unique IDs (e.g., FR1, NFR1). Functional requirements included user registration, event CRUD operations, admin moderation, and notifications. Non-functional requirements were made measurable (e.g., "95% of API responses < 500ms", "support 500+ concurrent users").

(i) Functional Requirements:

ID	Requirement
FR1	The system shall allow users to register and log in via email + password.
FR2	The system shall hash passwords with bcrypt (workFactor ≥ 10).
FR3	Authorized event creators shall create, edit, and delete events containing {title, description, date, time, city, state, category,

	imageURL, ticketURL, price, organizer}.
FR4	Public users shall browse, search, and filter events by city, state, date range, and category.
FR5	The system shall send email notifications on event creation, approval, or update using Nodemailer.
FR6	Admins shall approve/reject event creators and moderate events.
FR7	The system shall expose a shareable public URL for each event.

(ii) Non- Functional Requirements**:**

ID	Requirement	Metric & Target
NFR1	Performance	95 % API responses < 500 ms; p99 < 800 ms
NFR2	DB Search	Complex text/geo queries < 200 ms
NFR3	Concurrency	Support ≥ 500 simultaneous users with < 70 % CPU on t3.medium
NFR4	Availability	Uptime ≥ 99.5 % monthly (Pingdom)
NFR5	Security	OWASP Top-10 scan score A; 0 open high-sev CVEs
NFR6	Usability	SUS score ≥ 80 from 10 beta testers
NFR7	Portability	Fully containerized; deployable via docker-compose up on Linux/Mac/Win

Architecture & Technology:

The system uses a React.js frontend, Node.js/Express backend, and MongoDB Atlas for data storage. Authentication is handled via JWT, and images are managed with Cloudinary. The architecture was documented with a UML component diagram using standard notation. All technology choices were justified for scalability, maintainability, and developer productivity.

Testing Strategy:

Level	Framework / Tool	Target Coverage
Unit	Jest, Supertest (backend) / React Testing Library (frontend)	85 % backend, 75 % frontend
Integration	Supertest against staging container	CRUD flows, auth
E2E	Playwright	Happy paths on Chrome + Firefox
Manual Acceptance	Checklist below	5 key scenarios

(I) Manual Test Cases:

ID	Description	Preconditions	Steps / Input	Expected Result
TC-UI-01	Register new user	None	1. Navigate /register 2. Fill email alice@example.com, pwd Abc123!! 3. Submit	Success toast, redirected to /dashboard, JWT cookie set
TC-EV-02	Organizer creates event	User role =creator, approved	1. Click "Create Event" 2. Enter title = "Jazz Fest", city = "Boston", …	201 API, event appears in "Pending Approval"

TC-SR-03	Search by city & date	≥ 5 events exist	Filter city = “Boston”, date range “2025-07-01 … 2025-07-31”	≤ 200 ms, only Boston July events listed
TC-AD-04	Admin rejects event	Admin logged in	Open event id=42 → click “Reject”	Status -> rejected, creator email sent
TC-PF-05	Load test 500 users	Staging up	k6 script ramp to 500 VUs	95 % resp < 500 ms, no 5xx errors

Documentation:

Comprehensive documentation was maintained, including a README, API docs (Swagger/OpenAPI), UML diagrams, and an alphabetized glossary. All code and documentation were versioned in a public GitHub repository.

Lessons Learned

- Scrum is effective for managing uncertainty and incorporating feedback, but requires discipline in backlog refinement and sprint reviews.
- Explicit, measurable requirements (with IDs and metrics) are crucial for tracking progress and ensuring quality.
- Early and continuous testing (unit, integration, manual) helps catch issues before they escalate.
- Cloud-native tools (Docker, GitHub Actions, MongoDB Atlas) greatly simplify deployment and scaling, but require upfront learning.
- Clear documentation and diagrams (UML, API docs) are invaluable for onboarding and maintenance.
- Feedback channels (tutorials, peer reviews) are essential for improving the product and avoiding common pitfalls.

Resource Handling

Time: Sprints and daily standups kept the team focused and on schedule.

People: Defined Scrum roles ensured accountability and clear communication.

Tools: Trello for task management, GitHub for version control, and Postman for API testing streamlined the workflow.

Glossary

Term	Definition
Admin	user with rights to approve creators and moderate events
Backlog	ordered list of product work items maintained by the PO
Event Creator	authorized user who can add or manage events
Event Viewer	any public (unauthenticated) visitor browsing events
JWT (JSON Web Token)	stateless token used for authentication
Product Owner	Scrum role owning vision, backlog, ROI
Scrum Master	facilitator ensuring Scrum adherence
Sprint	fixed two-week iteration delivering "Done" increments

Conclusion

EventOnClick demonstrates the value of agile, test-driven, and cloud-native software engineering. The project met its functional and non-functional goals, and the lessons learned will inform future projects. All deliverables, including code, documentation, and test cases, are available in the public GitHub repository.