🎉 Event Management System - Production-Level Design

🔧 1. Core Features

A. Users

- User Registration / Login (OAuth, email/password, SSO)
- User Profiles
- Roles: `Admin`, `Organizer`, `Attendee`

B. Event Lifecycle

- Create/Update/Delete Event (Organizer)
- View/Search/Filter Events (Attendee)
- Event Types: Online, Offline, Hybrid
- Event Details: Date, Time, Venue, Speaker, Category, Tags

C. Ticketing

- Ticket Tiers: Free, Paid, VIP
- Seat Selection (for venue-specific)
- Ticket Limits per user
- QR Code generation for validation

D. Booking System

- Locking Mechanism for high-concurrency booking
- Waitlisting
- Booking History

E. Payment Integration

- Razorpay, Stripe, PayPal support
- Webhooks for payment success/failure
- Refund handling
- Coupons, Discounts

F. Notifications

- Email / SMS / In-app
- Reminders, Confirmations
- Integration: Twilio, SendGrid, Firebase

G. Admin Dashboard

- Event approval, moderation

- Reports and analytics
- Manage users, events, payments

⚙️ 2. Tech Stack

Component	**Choice**
Frontend	React.js / Next.js / Flutter
Backend	Node.js (Express/NestJS) or Go
API Gateway	NGINX / Kong / Express Proxy
DB (Main)	PostgreSQL
Caching	Redis
Search	Elasticsearch / MeiliSearch
File Storage	AWS S3 / Cloudinary
Messaging Queue	RabbitMQ / Kafka
Auth	JWT + OAuth2
Payments	Stripe / Razorpay
Notifications	Firebase, SendGrid, Twilio
Monitoring	Prometheus, Grafana, Sentry
Deployment	Docker + Kubernetes (EKS/GKE)
CI/CD	GitHub Actions / GitLab CI
Secrets Manager	AWS Secrets Manager / Vault

🧠 3. Microservices Breakdown

A. Auth Service

- JWT issuance & refresh
- Role-based Access Control
- OAuth2 (Google, Facebook)

B. User Service

- Profile Management
- Organizer Verification
- Attendee Preferences

C. Event Service

- CRUD on Events

- Search, Filter, Tags
- Pagination, Sorting

D. Booking Service

- Ticket allocation
- Locking (Redis + TTL)
- Seat assignment logic

E. Ticket Service

- QR Code generation (e.g., using `qrcode` lib)
- PDF ticket creation

F. Payment Service

- Order creation
- Webhook verification
- Refund handling

G. Notification Service

- Email templates
- SMS + Push notifications
- CRON jobs for reminders

H. Admin Service

- Approve/Deny events
- Report abusive events
- Analytics Dashboard

🔐 4. Security

- Rate Limiting (API Gateway or Redis)
- CSRF, XSS, SQLi protection
- HTTPS Everywhere
- 2FA for organizers
- Audit logs
- Role-based access at route and data level

📈 5. Scalability

- Use CDN for static assets
- Horizontally scalable services (K8s)
- Caching frequently queried data (Redis)
- Read replicas for PostgreSQL
- Use eventual consistency where possible (e.g., notifications)

- CQRS for Booking/Event read/write

📊 6. Observability

- Logging: ELK Stack or Loki
- Metrics: Prometheus + Grafana
- Error Monitoring: Sentry
- Health checks for each microservice
- Alerting with PagerDuty or Opsgenie

🧪 8. Testing Strategy

- Unit Tests (Jest, PyTest, etc.)
- Integration Tests (Postman + CI)
- Load Testing (k6, Locust)
- End-to-End Tests (Cypress)

🔄 9. CI/CD Pipeline

- Lint, Test, Build
- Docker Image build and push to ECR/GCR
- Auto deploy via Helm in Kubernetes
- Blue-Green deployment for rollback

📝 10. Database Design (Simplified)

`users`

```
id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
email TEXT UNIQUE NOT NULL,
password TEXT,
role TEXT CHECK (role IN ('admin', 'organizer', 'attendee')),
created_at TIMESTAMP DEFAULT NOW()
```

Events Table

```
id SERIAL PRIMARY KEY,
title TEXT NOT NULL,
description TEXT,
organizer_id INT REFERENCES users(id),
start_time TIMESTAMP,
end_time TIMESTAMP,
venue TEXT,
type TEXT CHECK (type IN ('online', 'offline', 'hybrid')),
category TEXT,
tags TEXT[]
```

Tickets Table

```
id SERIAL PRIMARY KEY,
event_id INT REFERENCES events(id),
type TEXT,
price NUMERIC,
total INT,
available INT
```

Bookings Table

```
id SERIAL PRIMARY KEY,
user_id INT REFERENCES users(id),
event_id INT REFERENCES events(id),
ticket_id INT REFERENCES tickets(id),
status TEXT CHECK (status IN ('pending', 'confirmed', 'cancelled')),
booked_at TIMESTAMP DEFAULT NOW()
```

Payments Table

```
id SERIAL PRIMARY KEY,
booking_id INT REFERENCES bookings(id),
status TEXT CHECK (status IN ('pending', 'success', 'failed', 'refunded')),
provider TEXT,
txn_id TEXT
```