India Got Latent - Design Document



Table of Contents

1. Project Overview

2. System Architecture

3. Database Schema

4. User Flows

5. API Structure

6. Security Considerations

7. Scalability Plans

8. User Interface Design

9. Testing Strategy

10. Deployment and DevOps

1. Project Overview

India Got Latent is a revolutionary talent show platform hosted by comedian Samay Raina. This web application manages the show's operations, participant registrations, event bookings, and implements a unique scoring system.

Key Features:

• Unique talent showcase

• Self-scoring system

• Multiple judge scoring

• Innovative winning mechanism

• Crowdfunded prize pool

• User authentication

• Participant management

• Event management

• Ticket booking system

• Collaborator and sponsor management

Target Audience:

• Participants with unique talents

• Judges (celebrities and industry experts)

• Audience members (ticket buyers)

• Sponsors and collaborators

Project Goals:

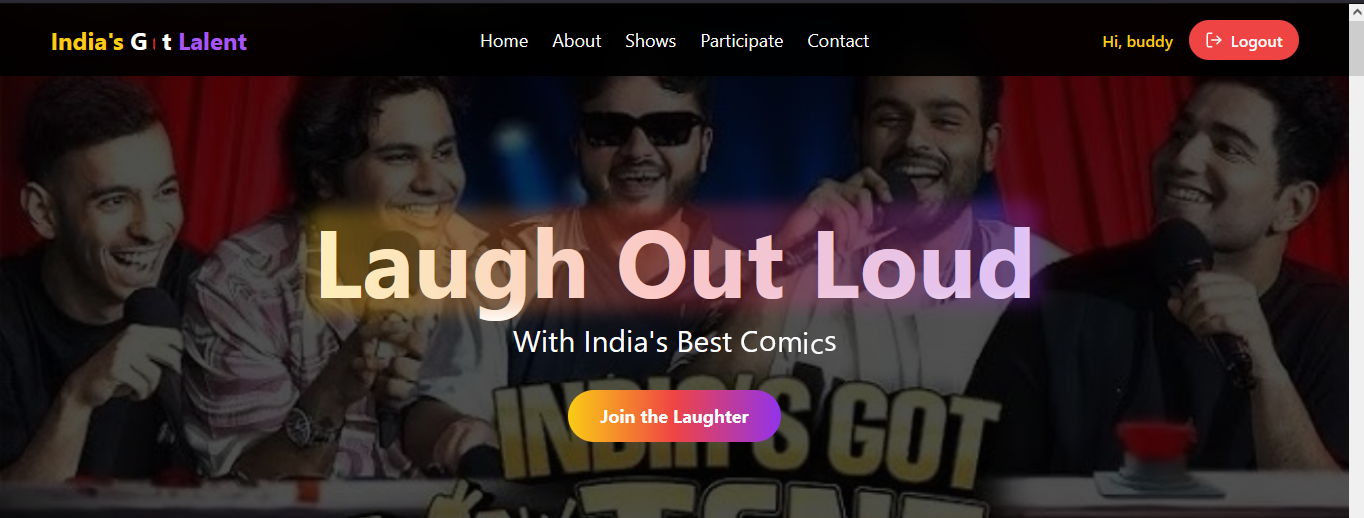
1. Create a platform that celebrates unconventional talents

2. Implement a fair and engaging scoring system

3. Provide an easy-to-use interface for all user types

4. Ensure scalability to handle high traffic during live events

5. Maintain data security and user privacy



2. System Architecture

India Got Latent follows a microservices architecture to ensure scalability and maintainability.

Component Descriptions:

1. Client: React.js with react vite framework

2. Load Balancer: Distributes incoming traffic

3. API Gateway: Routes requests to appropriate microservices

4. Microservices: User, Participant, Event, Booking, Scoring, Collaborator

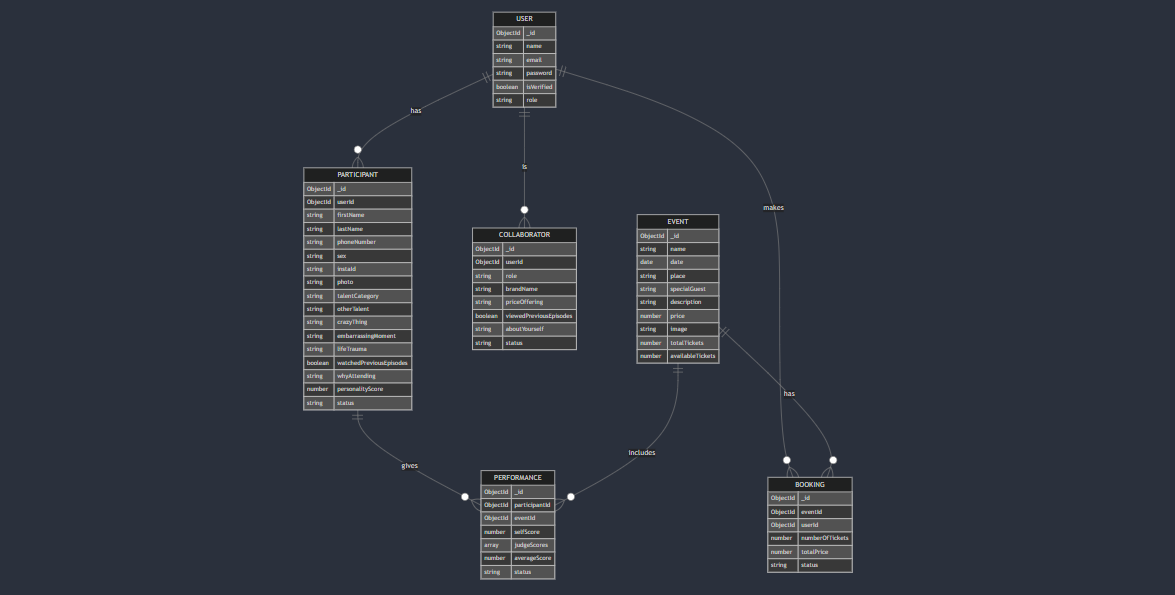
5. Databases: MongoDB for each microservice

6. Auth Service: Handles authentication and authorization

7. File Storage Service: Manages file uploads and retrieval

3. Database Schema

The database schema is designed to efficiently store and manage all aspects of the India Got Latent show.



4. User Flows

4.1 Participant Registration Flow

This Three step form includes the following steps:

1. Start

2. Visit Website

3. Click 'Register as Participant'

4. Fill Registration Form

5. Upload Photo

6. Submit Form

7. Form Valid? (If No, return to step 4)

8. Send Verification Email

9. Verify Email

10. Registration Complete

11. End

4.2 Ticket Booking Flow

The ticket booking include the following steps:

1. Start

2. Browse Events

3. Select Event

4. Choose Number of Tickets

5. Proceed to Checkout

6. User Logged In? (If No, Login/Register)

7. Enter Payment Details

8. Confirm Payment

9. Generate Ticket

10. Send Confirmation Email

11. End

5. API Structure

The API follows RESTful principles and is organized into the following main endpoints:

5.1 Authentication

• POST /api/auth/signup

• POST /api/auth/verify-email

• POST /api/auth/login

• POST /api/auth/logout

• POST /api/auth/refresh-token

5.2 Users

• GET /api/users

• GET /api/users/:id

• PUT /api/users/:id

• DELETE /api/users/:id

5.3 Participants

• POST /api/participants

• GET /api/participants

• GET /api/participants/:id

• PUT /api/participants/:id

• DELETE /api/participants/:id

5.4 Events

• POST /api/events

• GET /api/events

• GET /api/events/:id

• PUT /api/events/:id

• DELETE /api/events/:id

5.5 Bookings

• POST /api/bookings

• GET /api/bookings

• GET /api/bookings/:id

• PUT /api/bookings/:id

• DELETE /api/bookings/:id

5.6 Collaborators

• POST /api/collaborators

• GET /api/collaborators

• GET /api/collaborators/:id

• PUT /api/collaborators/:id

• DELETE /api/collaborators/:id

5.7 Performances

• POST /api/performances

• GET /api/performances

• GET /api/performances/:id

• PUT /api/performances/:id

• DELETE /api/performances/:id

6. Security Considerations

1. Authentication: JSON Web Tokens (JWT) are used for secure authentication.

2. Password Hashing: Passwords are hashed using bcrypt before storage.

3. HTTPS: All communications are encrypted using HTTPS.

4. Input Validation: Strict input validation is implemented on both client and server sides.

5. Rate Limiting: API rate limiting is implemented to prevent abuse.

6. CORS: Cross-Origin Resource Sharing is properly configured.

7. Environment Variables: Sensitive information is stored in environment variables.

8. SQL Injection Prevention: Use of ORM (Mongoose) helps prevent SQL injection attacks.

7. Scalability Plans

1. Horizontal Scaling: The microservices architecture allows for easy horizontal scaling of individual services.

2. Caching: Implement Redis caching for frequently accessed data.

3. Database Indexing: Proper indexing of MongoDB collections for optimized query performance.

4. Load Balancing: Utilize load balancing to distribute traffic across multiple servers.

5. CDN: Implement a Content Delivery Network for static assets.

6. Asynchronous Processing: Use message queues for handling time-consuming tasks asynchronously.

7. Database Sharding: Implement database sharding as data grows.

8. Serverless Functions: Utilize serverless functions for specific tasks to improve scalability and reduce costs.

9. Auto-scaling: Implement auto-scaling policies for cloud resources based on traffic patterns.

8. User Interface Design

8.1 Design System

The UI will follow a consistent design system with the following key components:

• Color Palette: Primary (#FF5733), Secondary (#33FF57), Accent (#3357FF)

• Typography: Headings (Montserrat), Body (Open Sans)

• Components: Buttons, Forms, Cards, Modals, Navigation

8.2 Key Screens

1. Home Page: Showcase upcoming events, featured talents, and call-to-action for registration

2. Participant Registration: Multi-step form for talent registration

3. Event Listing: Grid view of upcoming events with filtering options

4. Event Details: Detailed view of an event with ticket booking option

5. User Dashboard: Personalized dashboard for participants, judges, and audience

6. Live Scoring Interface: Real-time interface for judges to score performances

7. Leaderboard: Dynamic leaderboard showing top performers

8.3 Responsive Design

The UI will be fully responsive, ensuring a seamless experience across devices:

• Mobile-first approach

• Fluid layouts

• Adaptive images

• Touch-friendly interface elements

9. Testing Strategy

9.1 Unit Testing

• Use Jest for unit testing React components and JavaScript functions

• Aim for 80% code coverage

• Implement snapshot testing for UI components

10. Deployment and DevOps

10.1 CI/CD Pipeline

• Use GitHub Actions for continuous integration and deployment

• Implement automated testing in the CI pipeline

• Use Docker for containerization of microservices

10.2 Deployment Strategy

• Use Vercel for frontend deployment

• Deploy backend microservices to Kubernetes cluster on Google Cloud Platform

• Implement blue-green deployment for zero-downtime updates

10.3 Monitoring and Logging

• Use Prometheus for system monitoring

• Implement ELK stack (Elasticsearch, Logstash, Kibana) for centralized logging

• Set up alerts for critical system metrics and errors

10.4 Backup and Disaster Recovery

• Implement daily automated backups of all databases

• Set up geo-redundant storage for backups

• Develop and regularly test a disaster recovery plan

This comprehensive design document provides a detailed overview of the India Got Latent project, covering all aspects from system architecture to deployment strategies. It serves as a guide for development, ensuring all team members are aligned on the project's goals, structure, and implementation details.