Tab 1

EVENT QUEST

1. Project Overview

Goal of the Project:

The goal of this project is to build a scalable and user-friendly **online event handling platform** using Django, a Python web framework. The platform will allow users to create, manage, and participate in online events such as webinars, workshops, or conferences. It will include features such as event creation, registration, payment processing, reminders, and real-time event updates. The project will **not include** third-party event hosting (like video conferencing) or advanced analytics/reporting for events, which will be considered in future versions.

Importance :

The project is important as it addresses the increasing need for virtual event management solutions, especially in a post-pandemic world where remote interaction has become more critical. Specific goals or outcomes include:

1. **Seamless Event Creation and Management:** Organizers can create and manage events with an easy-to-use interface.
2. **User Registration & Participation:** Attendees can register for events and receive confirmations, reminders, and links.
3. **Payment Integration:** For paid events, secure payment options will be integrated.
4. **Scalability:** The platform will be designed to handle high volumes of events and participants efficiently.
5. **Real-time Notifications:** Both organizers and participants will receive live updates on the status of events, schedule changes, and announcements.

Team Members:

Team Leader: Shanky Kumar
Team Members: 1. Sahithi Pokuri
2. Sonali Singh

3. Sanika Desai

Role of Team Members:
Shanky - Backend for Module 1 and 3, Frontend for Module 2
Sahithi - Frontend for Module 1 and 3, Backend for Module 2
Sanika - Backend for Module 1 and 3, Frontend for Module 2
Sonali - Frontend for Module 1 and 3, Backend for Module 2

Frontend - HTML, CSS, JavaScript
Backend - Python Django

2. Requirements Documentation

Core Features and Functions:

- ***User Registration and Login:*** Users should be able to create accounts and securely log in to book appointments and manage their bookings.
- ***Hall Selection:*** Users can view available halls with details on capacity, amenities, and pricing.
- ***Appointment Scheduling:*** Users should be able to specify a date, time slot, and duration for their booking, and submit these details through a form.
- ***Appointment Viewing:*** Users should be able to view their already booked appointments and check it's status.
- ***Availability Check:*** The system should verify the availability of a selected hall in real time based on the user's selected date and time.
- ***Booking Confirmation***: After a successful booking, users should receive a confirmation email or notification.
- ***Manage Bookings***: Users can view, edit, or cancel their bookings as needed.
- ***Admin Management:*** Administrators can view all bookings, manage hall schedules, and approve or deny bookings as necessary.

Performance and Quality standards:

Performance:

- The application should respond to user inputs (e.g., booking form submission) quickly, ideally within 1–2 seconds.

- Real-time availability checks should be fast to avoid delays and improve user experience.

Security:

- User authentication and authorization should be implemented to protect user data and prevent unauthorized access.
- Data privacy should be ensured, especially for sensitive information such as contact details and email addresses.
- The system should prevent double booking by securing the availability check and booking process.

Usability:

- The interface should be intuitive and easy to navigate for users with minimal technical skills.
- Clear error messages should guide users if they encounter an issue (e.g., trying to book an unavailable time slot).
- Responsive design is necessary so the application is usable on both desktop and mobile devices.

Reliability:

The system should be highly reliable with minimal downtime, especially during high-traffic periods.

Scalability:

The application should be able to handle an increasing number of bookings and user traffic without significant performance degradation.

Business Objectives:

- ***Enhanced Customer Service:*** Providing a convenient, accessible, and efficient way for users to book halls and appointments aligns with the goal of improving customer service.
- ***Revenue Growth:*** By streamlining the booking process, the application encourages more users to book spaces, contributing to potential revenue growth.

- ***Operational Efficiency:*** Automating booking and management reduces the need for manual intervention and minimizes scheduling errors.
- ***Brand Differentiation:*** Having a user-friendly online booking system sets the organization apart from competitors who may rely on less convenient booking methods.

Alignment with Company Goals:

- The project aligns with company goals of digital transformation by introducing a modern, customer-oriented solution.
- It demonstrates the company's commitment to embracing technology to improve user engagement and satisfaction.

User side interaction:

User Stories:

- As a user, I want to easily view available halls so that I can select the one that best suits my event's needs.
- As a user, I want to check the availability of a hall on a specific date and time so that I can book without conflicts.
- As a user, I want to receive a confirmation for my booking so that I have proof of my appointment details.
- As an admin, I want to view all upcoming bookings so that I can manage hall usage effectively.
- As an admin, I want to approve or deny booking requests based on availability and other conditions so that the scheduling system remains organized and fair.

Use Cases:

- *Booking a Hall:* The user logs in, selects a hall, specifies a date and time, and submits the booking form. The system checks availability and confirms the booking if the hall is free.
- *Modifying a Booking:* The user views their current bookings, selects a booking they want to modify, updates the details, and submits. The system checks if the updated time is available and confirms the changes.

- *Viewing Appointments:* The admin logs in, views a dashboard showing all scheduled appointments, and can filter by hall, date, or user to monitor and manage bookings.

3. Project Plan

When will it be done?

A high-level timeline for the project with key phases and milestones:

Phase	Milestone	Timeline
Planning	Requirements gathering, scope definition, and design blueprint completed.	Week 1
Development	Initial prototype with core features implemented (like hall booking and appointment scheduling).	Weeks 2 - 4
Testing	Functional, UI/UX, and performance testing completed; bugs identified and fixed.	Week 5
Deployment	Finalized system deployed for end-user access.	Week 6
Post - Deployment	System monitoring, feedback collection, and minor updates as required.	After week 6

What do we need?

Resources required for successful completion of the project:

1. **People:**
    - Project Manager: Oversees planning and delivery.
    - Backend Developer: Builds Django-based APIs and logic.
    - Frontend Developer: Implements HTML/CSS and integrates with backend.
    - QA Engineer: Tests features for functionality and usability.

- UI/UX Designer: Designs intuitive user interfaces.

2. **Equipment:**
    - Development machines (laptops/desktops) with required software installed.
    - Servers or cloud infrastructure for deployment (e.g., AWS, Azure).

3. **Software:**
    - ***Development:*** Python, Django, HTML, CSS, and supporting libraries.
    - ***Version Control:*** Git and GitHub/GitLab.
    - ***Testing:*** Selenium for UI tests, Postman for API tests.
    - ***Deployment:*** Nginx/Apache for web server setup.

How much will it cost?

An estimated budget for the project:

Category	**Cost Estimate**
Development	₹4,15,000
UI/UX Design	₹83,000
Testing	₹66,400
Deployment	₹41,500
Miscellaneous	₹58,100
Total Estimated Cost	₹6,64,000

This budget ensures that all aspects of the project, from development to deployment, are adequately covered.

4. Architecture and design documentation

How is it built?

The system is built using a client-server architecture:

1. **Frontend(Client):**
   - ***Technologies:*** HTML, CSS for structure and styling.
   - ***Functionality:*** Provides the user interface for customers to interact, such as booking appointments, managing profiles, and viewing schedules.

2. **Backend(Server):**
   - ***Technologies:*** Django for handling business logic, processing requests, and connecting to the database.
   - ***APIs:*** Responsible for CRUD operations like saving, fetching, and updating appointment or profile data.

3. **Database:**
   - ***Technology:*** SQLite or MySQL for storing all user, booking, and hall-related data.
   - ***Interaction:*** Handled by Django ORM.

4. **Subsystems:**
   - ***Authentication:*** Ensures secure login/logout and user session handling.
   - ***Appointments Management:*** Handles the booking and viewing of appointments.
   - ***Hall Management:*** Manages hall availability and booking.
   - ***Profile Management:*** Enables users to update personal details and track their activities.

5. Testing and Quality Assurance

How will we ensure it works?

Our testing approach will be thorough and systematic, ensuring the online event management system operates as expected under various scenarios. The types of tests we will conduct include:

- **Unit Tests**: Focus on testing individual components, such as models, views, and utility functions, using tools like `pytest` or Django's built-in testing framework.
- **Integration Tests**: Verify that different modules of the system (e.g., user authentication and event creation) interact correctly.
- **System Tests**: Test the complete application in an environment that mimics production, covering user workflows from start to finish.
- **Performance Tests**: Ensure the system handles a high number of simultaneous users effectively.
- **Security Tests**: Identify vulnerabilities in authentication, data handling, and permissions.

Specific Test Cases

1. **Unit Tests**:
    - Validating form inputs for event creation and registration.
    - Testing permissions for different user roles (admin, organizer, attendee).
2. **Integration Tests**:
    - User authentication flows (e.g., login, registration, password reset).
    - Event creation and ticket purchase processes.
3. **System Tests**:
    - A user registers, logs in, creates an event, and sees it listed.
    - Multiple users register for the same event without conflicts.
4. **Security Tests**:
    - Ensure sensitive data (e.g., passwords) is encrypted.
    - Prevent unauthorized data access using penetration testing tools like OWASP ZAP.

What makes it complete?

The project will be considered finished and accepted by stakeholders when the following conditions are met:

- All specified features are implemented and tested successfully.
- All high-priority bugs are resolved, and no critical bugs remain.
- Performance metrics meet the predefined benchmarks (e.g., response time under high load).
- Security vulnerabilities are addressed, and the application passes security audits.
- Stakeholders approve the final demonstration of the system, confirming it aligns with their requirements.

When will we test?

- **Development Phase**: Unit tests will run after each feature implementation as part of the CI/CD pipeline.
- **Integration Testing Phase**: After completing feature modules, integration tests will verify component interaction.
- **Pre-Deployment Phase**: Comprehensive system and performance testing will occur in a staging environment.
- **Post-Deployment Phase**: Ongoing monitoring and regression tests to address updates or feature additions.
- **CI/CD**: Automated testing will be incorporated into the CI/CD process using tools like Jenkins, GitHub Actions, or GitLab CI.

What if something goes wrong?

Issues will be managed as follows:

- **Reporting and Tracking**: Use GitHub Issues or JIRA to log bugs, categorize them (e.g., critical, high, low), and track progress.
- **Fixing**: Developers will work on issues based on priority, with critical bugs addressed immediately.
- **Validation**: Once resolved, each issue will undergo regression testing to ensure the fix doesn’t introduce new bugs.

6. Deployment and Implementation plan

Where will it live?

The system will be deployed on a **cloud environment** such as AWS, Azure, or Google Cloud. The specific services may include:

- **Django Application**: Deployed using AWS Elastic Beanstalk or Docker containers in Kubernetes.
- **Database**: Hosted using managed services like AWS RDS or Google Cloud SQL for scalability.
- **Static Files**: Served via a CDN (e.g., AWS CloudFront) to improve performance.

How will we get it there?

Deployment will follow a **phased rollout strategy**:

1. **Staging Environment**: All features and fixes are deployed here for final validation.
2. **Production Environment**: Deployment occurs after stakeholder approval, using tools like `Ansible`, `Terraform`, or cloud provider pipelines.

What if something goes wrong?

1. **Rollback Strategy**: Deployments will include versioning to allow rollback to the last stable version in case of critical failures.
2. **Monitoring and Alerts**: Tools like New Relic or CloudWatch will be used to monitor performance and send alerts if anomalies are detected.
3. **Backups**: Automated daily backups of the database and configurations will be set up to prevent data loss.

How will users learn to use it?

1. **User Training**:
    - Online documentation covering key workflows like event creation and registration.

    - Video tutorials and step-by-step guides accessible through the system's Help section.
2. **Onboarding**:
    - A guided onboarding process for first-time users.
    - FAQs and chat support for immediate assistance.

This approach ensures a smooth deployment and effective adoption of the system.

7. Maintenance and Support

Who's in charge of keeping it running?

The **IT Support Team**, consisting of developers and system administrators, is responsible for ongoing support and maintenance of the **Online Event Management System**.

- ***Developers:*** Handle bug fixes, feature enhancements, and software updates.
- ***System Administrators:*** Manage server uptime, backups, and infrastructure performance.

What needs to be done to keep it running?

Regular maintenance tasks include:

1. ***Software Updates:*** Apply security patches and update Python, Django, and third-party libraries.
2. ***Bug Fixes:*** Address reported issues promptly.
3. ***Performance Monitoring*****:** Use tools like New Relic or Grafana to monitor system health and performance metrics.
4. ***Database Maintenance:*** Regular optimization and backup of the database to prevent data loss or corruption.
5. ***Server Maintenance:*** Ensure server resources (CPU, memory, disk space) are adequate and scalable for system demand.
6. ***Security Audits:*** Periodic reviews of application and server configurations to address vulnerabilities.

How will we know what users think?

1. ***User Feedback Mechanisms:*** Integrate feedback forms and surveys into the platform for user input.
2. ***Support Channels:*** Use email, a support ticketing system, or live chat for issue reporting.
3. ***Usage Analytics:*** Implement tools like Google Analytics or Mixpanel to track user behavior and identify pain points.
4. ***Regular Review Meetings:*** Analyze feedback monthly and prioritize actionable changes for development.

What are our service commitments?

1. ***Uptime Guarantee:*** Maintain 99.9% system uptime with scheduled maintenance announced in advance.
2. ***Response Times:***
    - Critical issues: Response within 1 hour, resolution within 4 hours.
    - High-priority issues: Response within 4 hours, resolution within 1 business day.
    - General queries: Response within 24 hours, resolution within 3 business days.
3. ***Data Recovery:*** Restore lost or corrupted data within 2 hours of detection.

8. Risk Management

What could go wrong?

1. **Technical Risks:**
    - Server outages due to unexpected demand or cyberattacks.
    - Bugs or vulnerabilities in the codebase.
    - Data loss or corruption during updates.
2. **Operational Risks:**
    - Lack of skilled personnel for urgent fixes.
    - Delays in addressing user-reported issues.
3. **Financial Risks:**
    - Budget overruns in hosting, support, or third-party tools.

How can we prevent problems?

1. ***Server Redundancy:*** Implement load balancing and failover mechanisms.
2. ***Automated Testing:*** Perform rigorous unit, integration, and regression testing before deployments.
3. ***Monitoring Tools*****:** Use real-time monitoring to identify and resolve performance issues early.
4. ***Training*****:** Ensure team members are well-trained in Django and system operations.
5. ***Budget Planning:*** Allocate funds for emergencies and unexpected expenses.

What's our backup plan?

1. ***Data Backups:*** Regular, automated backups stored securely offsite.
2. ***Alternative Workflows:*** Temporary shutdown of advanced features while maintaining basic functionality during critical issues.
3. **Incident Management:** Follow a predefined incident response plan, including communication protocols for notifying users and stakeholders.

Who's watching for problems?

The **Risk Management Team** is responsible for identifying, tracking, and managing risks. This team comprises:

- A ***Project Manager***, overseeing risk tracking and mitigation strategies.
- ***Developers***, monitoring technical issues and deploying fixes.
- ***System Administrators***, ensuring server and data integrity.

9. Security And Privacy

Data Protection

a. Encryption:

- At Rest: Use database encryption techniques to protect sensitive data stored in the system, such as user credentials, payment details, and personally identifiable information (PII). Tools like Django's support for encrypted fields can help achieve this.

- In Transit: Enable HTTPS using SSL/TLS certificates to secure data transmitted between clients and servers.
- Password Hashing: Store user passwords securely using Django's built-in hashing system (e.g., PBKDF2 or Argon2).

b. Access Controls:

- Implement role-based access control (RBAC) to restrict access based on user roles (e.g., admin, organizer, attendee).
- Use Django's authentication and permissions framework to enforce fine-grained access controls.
- Regularly review and update user permissions to prevent unauthorized access.

c. Compliance with Data Privacy Regulations:

- Ensure compliance with laws like GDPR (General Data Protection Regulation) and HIPAA (Health Insurance Portability and Accountability Act) if handling EU residents' data or healthcare data.
    - Allow users to access, modify, or delete their personal data upon request (GDPR compliance).
    - Encrypt health-related information to meet HIPAA requirements.
    - Maintain a privacy policy document explaining how user data is collected, used, and stored.

Security Measures

a. Firewalls:

- Use **application firewalls** (e.g., Django REST Framework's throttling and rate-limiting mechanisms) to prevent excessive requests and brute-force attacks.
- Deploy network firewalls (e.g., AWS Security Groups) for server-level protection.

b. Intrusion Detection and Prevention:

- Monitor the application for suspicious activities using tools like **OSSEC**, **Snort**, or cloud monitoring solutions (e.g., AWS GuardDuty, Azure Security Center).
- Integrate with Django's **logging framework** to log security-relevant events and analyze them regularly.

c. Regular Security Audits:

- Perform **penetration testing** to identify vulnerabilities in the application.
- Run automated security tools like **OWASP ZAP** and **Bandit** to check for security flaws in the codebase.
- Keep all dependencies (e.g., Django, third-party libraries) updated to address known vulnerabilities (use tools like **pip-audit**).

d. Secure Deployment:

- Use secure configurations for deployment environments, such as disabling debug mode (`DEBUG = False`) and restricting allowed hosts (`ALLOWED_HOSTS`).
- Implement **Content Security Policies (CSP)** to prevent cross-site scripting (XSS) attacks.

Incident Response Plan

a. Detection:

- Set up alerts for abnormal behavior, such as multiple failed login attempts or unexpected changes in database entries.
- Use tools like **Sentry** to detect and report application errors.

b. Containment:

- Quickly isolate affected systems or services to prevent further damage.
- Implement backup policies to restore data and services without major downtime.

c. Eradication:

- Identify the root cause of the incident (e.g., malware, outdated dependency) and remove it.
- Patch vulnerabilities immediately and verify that no backdoors remain in the system.

d. Recovery:

- Restore services from backups after ensuring they are free of vulnerabilities.
- Test the system thoroughly before making it available to users.

e. Post-Incident Review:

- Conduct a **post-mortem analysis** to document the incident, what went wrong, and how it was resolved.
- Update security protocols and provide training to team members to avoid similar incidents in the future.

10. Legal And Compliance

Licensing

Licensing ensures the legal use of software and hardware components within the project.

a. Software Licenses:

- Python and Django Framework:
    - Python is distributed under the PSF License, and Django under the BSD License, both permissive open-source licenses that allow modification and distribution with minimal restrictions.
- Third-party Libraries:
    - Check the licenses of third-party libraries used in the project (e.g., MIT, Apache, or GPL licenses). Some libraries might have restrictions, such as requiring attribution or prohibiting proprietary use.
    - Use tools like pip-licenses to generate a report of dependencies and their licenses.

b. Frontend and UI Components:

- If using templates, fonts, or design components (e.g., Bootstrap or FontAwesome), ensure compliance with their respective licenses (usually open-source but may have premium options).

c. Hosting and Database Services:

- Ensure licensing agreements for hosting platforms (e.g., AWS, Azure) and databases (e.g., PostgreSQL, MySQL) are met.

d. Hardware and Cloud Services:

- If physical servers are used, ensure compliance with server hardware warranties and licensing agreements for system software.

Regulatory Compliance

Regulatory compliance ensures the system meets the standards of the industry and the legal jurisdictions where it operates.

a. Data Privacy Laws:

- **GDPR (General Data Protection Regulation):**
  - Applicable if the system collects data from EU residents. Implement data minimization, user consent, and mechanisms to allow users to manage their data (e.g., view, update, delete).
  - Add a clear privacy policy and cookie consent mechanisms.
- **CCPA (California Consumer Privacy Act):**
  - Similar to GDPR, but applicable to California residents. Ensure mechanisms for disclosing data usage and honoring opt-out requests.
- **HIPAA (Health Insurance Portability and Accountability Act):**
  - If the platform handles health-related data (e.g., event management for healthcare conferences), implement stringent encryption and access controls to safeguard patient data.

b. Accessibility Standards:

- Adhere to **WCAG (Web Content Accessibility Guidelines)** to ensure the platform is accessible to users with disabilities.
- Use tools like Django-accessibility plugins to enhance compliance.

c. Taxation Compliance:

- If the system includes payment processing for events, ensure it complies with tax laws in the jurisdictions where transactions occur, such as **VAT (Value-Added Tax)** or **GST (Goods and Services Tax)**.

d. Industry-Specific Standards:

- If the platform caters to specific industries (e.g., medical, finance), identify and comply with industry regulations like **FDA (Food and Drug Administration)** or **ISO 27001** for information security.

Intellectual Property

Protecting intellectual property ensures that the project's innovations and branding are legally safeguarded.

a. Copyrights:

- Automatically protect the original source code, design elements, and documentation under copyright law.
- Add copyright notices in the codebase (e.g., `© [Year] [Company Name]`) and use software licenses (e.g., MIT or Apache) for your code if it will be shared.

b. Trademarks:

- Trademark the name, logo, and branding of the online event management system to prevent unauthorized use by competitors.
- Conduct a trademark search to ensure the selected name and logo are not already in use.

c. Patents:

- If the project includes innovative features (e.g., a novel algorithm for event scheduling), consider filing for a patent to protect these innovations.
- Consult a patent attorney to ensure eligibility and prepare necessary documentation.

d. Third-party Content:

- For images, videos, and other media, ensure proper licensing if sourced from external providers (e.g., Creative Commons or royalty-free sites).

e. Contributor Agreements:

- If the project involves contributions from multiple developers or freelancers, draft **Contributor License Agreements (CLAs)** to clarify ownership and usage rights for all contributions.

11. Environmental Impact Assessment

Sustainability

a. Energy Consumption:

- Efficient Coding Practices:

    - Optimize the Django application by writing efficient queries, reducing redundant computations, and caching frequently accessed data to reduce server workload and energy use.
    - Use tools like **Django Debug Toolbar** to identify and address performance bottlenecks.
- Server Optimization:
    - Use **auto-scaling** to dynamically allocate server resources based on demand. For instance, during low traffic periods, reduce the number of active servers to conserve energy.
    - Choose hosting providers with a focus on energy efficiency, such as those powered by renewable energy (e.g., Google Cloud, AWS with renewable energy regions).

b. Waste Generation:

- Paperless Operations:
    - Promote paperless event management by offering digital tickets, invoices, and certificates to reduce physical waste.
    - Encourage event organizers and attendees to use the platform for virtual schedules, announcements, and feedback collection.
- Longevity of Hardware:
    - If managing physical servers, extend hardware life through regular maintenance and upgrades rather than frequent replacements.

Green IT Practices

a. Energy-Efficient Hardware:

- Cloud Hosting Services:
    - Opt for energy-efficient cloud hosting platforms. Providers like AWS, Google Cloud, and Microsoft Azure operate energy-efficient data centers, often using renewable energy sources and advanced cooling systems.
- Hardware Consolidation:
    - Consolidate multiple workloads on fewer physical machines using virtualized infrastructure or containerization tools like Docker to reduce hardware energy demands.

b. Virtualized Infrastructure:

- Virtual Machines and Containers:
    - Use containerization (e.g., Docker) to deploy the Django application efficiently. Containers reduce resource overhead compared to traditional virtual machines.
    - Implement Kubernetes for managing containers, ensuring optimal resource allocation and scaling.

c. Efficient Resource Allocation:

- Use serverless architectures (e.g., AWS Lambda) for handling specific backend tasks. Serverless computing ensures that resources are only consumed when needed, minimizing idle time and energy use.

d. Sustainable Hosting Providers:

- Select hosting providers that support carbon-neutral operations. Some platforms participate in global sustainability programs or invest in renewable energy credits to offset carbon emissions.

e. Recycling and Disposal:

- For any unavoidable physical hardware use, adopt responsible e-waste disposal practices. Partner with certified e-waste recyclers to ensure safe and environmentally friendly recycling of outdated equipment.

f. Renewable Energy:

- Encourage event organizers to use renewable energy for physical events. The platform can include features to showcase green-certified venues or suggest eco-friendly vendors.

12. User Documentation

User Manuals:

The Event Management System allows users to book halls, schedule appointments, check availability, and manage events efficiently through a user-friendly interface.

Page-Specific Guides:

Each page has dedicated instructions in the manual:

- **Login and Registration:**
  Navigate to the 'Register' page, fill in the required details, and click 'Submit'. Use the 'Login' page to access your account with the registered credentials.
- **Book Hall:**
  Go to the 'Book Hall' page, fill in the event details, and click 'Submit'. You will be redirected to the availability check page for confirmation.
- **Check Hall Availability:**
  Use the 'Check Hall Availability' page to verify if your desired hall and date are available.
- **Appointments:**
  Submit appointment details on the 'Book Appointment' page and view all your bookings under 'View Appointments'.

Troubleshooting Tips:

If your booking form isn't submitting, ensure all mandatory fields are completed and valid.

User Interface Design:

The application's user interface has been thoughtfully designed to prioritize ease of use:

- **Navigation:**
  The top navigation bar provides direct links to all major features:
  - *Login, Register, Book Hall, Check Hall Availability, Book Appointment, View Appointments.*

- **Responsive Design:**
  The layout is fully responsive and accessible across devices, ensuring a consistent experience on desktops, and laptops.
- **Visual Hierarchy:**
  Key sections are highlighted for quick recognition. For example:
    - The banner prominently displays the application's name and tagline: *"Event Quest - Co-operate Event Management".*
- **Form Validations:**
  Booking and registration forms include client-side and server-side validation to prevent errors and improve user experience.
- **Accessibility Features:**
    - All links and buttons have descriptive labels.
    - High contrast and hover effects help visually differentiate elements for accessibility.

13. Project Management and Monitoring

Project Planning:

A detailed project plan was developed to ensure efficient execution of the Event Management System Application, covering the following aspects:

- **Tasks and Deliverables:**
    - Requirements Gathering: Identifying user needs, system features, and database structure.
    - Development: Building features like user registration, hall booking, availability checks, and appointment management.
    - Frontend Design: Designing an intuitive and visually appealing interface using HTML and CSS.
    - Backend Implementation: Using Django to handle user authentication, database management, and email notifications.
- **Timelines:**
  Each phase was allocated a specific timeline to ensure timely completion:

    - Week 1: Requirements analysis and database schema design.
    - Week 2–3: Development of core functionalities and UI design.
    - Week 4: Integration of email notifications and additional features.
    - Week 5: Testing and debugging.
    - Week 6: Final review
- **Resource Allocation:**
    - Tools and Technologies: Django, MySQL, HTML/CSS.
    - Team Members: Allocated roles for frontend development, backend implementation, and testing.
    - Infrastructure: Local development setup

Risk Management:

Potential risks were identified during the project and corresponding mitigation strategies were implemented:

- **Risk 1: Delayed Development Due to Complexity**
    - *Mitigation Strategy:* Tasks were divided into smaller, manageable modules with clearly defined objectives and deadlines.
- **Risk 2: Database Connectivity Issues**
    - *Mitigation Strategy:* The database connection was tested early in the development cycle, and a fallback local database was configured for testing purposes.
- **Risk 3: Email Notification Errors**
    - *Mitigation Strategy:* Configured SMTP settings with thorough testing and debug logging to handle potential email delivery issues.
- **Risk 4: Deployment Challenges**
    - *Mitigation Strategy:* Comprehensive documentation of deployment steps for Render was created. A staging environment was used for testing the live deployment before making it public.
- **Risk 5: User Interface Usability Issues**
    - *Mitigation Strategy:* Regular feedback was gathered from stakeholders during development to refine the UI design and ensure an intuitive experience.
- **Risk 6: Security Vulnerabilities**
    - *Mitigation Strategy:*

    - Sensitive data like the `SECRET_KEY` was secured using environment variables during deployment.
    - User authentication and authorization mechanisms were implemented to protect user data.

14. Testing And Quality Assurance

Unit Testing:

Unit testing focused on ensuring that individual components of the **EventQuest**functioned correctly. Each module was tested in isolation for functionality and reliability.

- **Login and Registration Module:**
    - Verified that users could register, log in, and log out successfully.
    - Tested validation rules for input fields, such as email format and password strength.
- **Hall Booking Feature:**
    - Ensured that users could select and submit booking details without errors.
    - Tested form validation for required fields (e.g., date, hall size, purpose).
- **Email Notification Functionality:**
    - Tested that email notifications were sent successfully after booking confirmation and user registration.
- **Availability Check Module:**
    - Verified that the system correctly identified whether a hall was available for the selected date and time.
- **Appointment Management:**
    - Checked that users could book appointments, view upcoming appointments, and edit or cancel them.

Integration Testing:

Integration testing was conducted to ensure seamless interaction between components and consistent data flow across the application:

- **Frontend-Backend Integration:**
    - Verified that the Django backend processed requests from the frontend and returned the correct data.
    - Tested API endpoints for data retrieval and updates.
- **Database Interaction:**
    - Ensured proper storage and retrieval of user, booking, and appointment details from the MySQL database.
    - Confirmed that foreign key relationships were maintained and data integrity was preserved.
- **Email Notification and Booking System:**
    - Tested the integration of the SMTP email service with the booking and registration modules.
- **Static and Media File Management:**
    - Verified that static assets (CSS, images) and media files (e.g., uploaded user documents) were correctly handled in production environments.