Competition Management System

The Competition Management System is designed to facilitate the management of various competitions across multiple categories.

Team

- Brian Kipkoech – Front End
- Kevin kiprop –Front End
- Moses Ogada – Back End

These roles were decided based on the area of expertise each one of us was to choose during specialization and the level of skill each of us felt comfortable with for the best results.

Technologies

We opted to choose HTML, CSS and Javascript for the Front End and Python languages for the Back End. MySQL was to be used for the database and we had to connect the Front End and the Back End using RESTful API.

Django was the framework we opted for instead of flask for the back end and we had to integrate the React.js for the Front End because it is a JavaScript library and we opted to use JavaScript for the Front End.

Challenge

The project is intended to automate, manage risk, and make work easy of Judges, Participants and Institutional Representatives during competitions. For a long time this has always been done manually which was time consuming, prone to errors, inefficient and costly.

The project may not be able to solve cheating, corruption experienced during competitions, and they cannot replace the nuanced decision-making and innovation that come from human insight.

The project is to be used by the Administrator, the Judges, Institutional Representatives and the Participants. The Judges evaluate and views projects. The Participants view the evaluation of the projects and download certificates after judges have done evaluation. The Institutional Representative registers the projects and participants. The Administrator customizes the categories and registers the judges.

Risks

Technical risks:

1. Data Security and Privacy

Risks:

- Unauthorized access to sensitive data (personal information of participants, scores, etc.).
- Data breaches resulting from vulnerabilities in the system.

Impact:

- Loss of trust from users and stakeholders.
- Legal consequences due to non-compliance with data protection regulations (e.g., GDPR).

Safeguards/Alternatives:

- Implement robust encryption for data at rest and in transit.
- Regularly update and patch the system to protect against known vulnerabilities.
- Use secure authentication mechanisms (e.g., multi-factor authentication).
- Conduct regular security audits and penetration testing.
-

2. System Reliability and Availability

Risks:

- Downtime during critical periods (e.g., submission deadlines, live announcements).
- Performance issues under high load (e.g., many users accessing the system simultaneously).

Impact:

- Frustration and dissatisfaction among users.
- Potential loss of participants or reputation damage.

Safeguards/Alternatives:

- Design for scalability using cloud services that can handle variable loads.
- Implement load balancing and failover mechanisms.
- Conduct thorough load testing before critical events.
- Monitor system performance in real-time and have a quick-response plan for outages.

3. Data Integrity and Accuracy

Risks:

- Errors in data processing (e.g., incorrect scores, participant information).
- Corruption or loss of data.

Impact:

- Unfair competition outcomes.
- Loss of credibility and potential disputes or legal issues.

Safeguards/Alternatives:

- Use transactional databases with ACID properties to ensure data integrity.
- Implement thorough validation checks and data verification processes.
- Regularly back up data and have a disaster recovery plan in place.

4. User Experience and Accessibility

Risks:

- Poor interface design leading to user confusion or errors.
- Lack of accessibility features for users with disabilities.

Impact:

- Reduced participation and engagement.
- Negative feedback and potential exclusion of certain user groups.

Safeguards/Alternatives:

- Conduct usability testing with real users to identify and fix issues.
- Follow best practices in UI/UX design, ensuring the system is intuitive and user-friendly.
- Implement accessibility standards (e.g., WCAG) to make the system inclusive.

5. Compliance with Regulations and Standards

Risks:

- Non-compliance with relevant legal and industry standards.
- Overlooking jurisdiction-specific regulations (e.g., different rules for different countries).

Impact:

- Legal penalties and fines.
- Restrictions on the operation of the system in certain regions.

Safeguards/Alternatives:

- Consult with legal experts to ensure compliance with applicable laws.
- Stay updated on regulatory changes and adapt the system as needed.
- Implement comprehensive documentation and auditing processes.

6. Integration with External Systems

Risks:

- Incompatibility or failures in integrating with third-party systems (e.g., payment gateways, social media platforms).
- API changes or discontinuation by external providers.

Impact:

- Disruption of key functionalities (e.g., payment processing, user authentication).
- Increased maintenance effort to manage integrations.

Safeguards/Alternatives:

- Use well-documented and widely adopted APIs for integrations.
- Regularly monitor and test integrations to ensure they function correctly.
- Have contingency plans and alternative providers in case of API changes or discontinuation.

7. Scalability and Maintainability

Risks:

- The system becoming difficult to scale as the number of users and competitions grow.
- High maintenance costs and complexity over time.

Impact:

- Inability to support growth and increased demand.
- Increased operational costs and technical debt.

Safeguards/Alternatives:

- Adopt modular and micro services architecture to facilitate scaling and maintenance.
- Use automated deployment and continuous integration/continuous deployment (CI/CD) pipelines.
- Document code thoroughly and follow coding standards to ensure maintainability

Competition Management System

Non-technical risks:

1. Stakeholder Misalignment

Risks:

- Conflicting expectations and requirements from different stakeholders (organizers, sponsors, participants

Impact:

- Project delays and increased costs due to constant changes and rework.
- Decreased satisfaction and support from key stakeholders.

Strategies:

- Engage stakeholders early and regularly through meetings, surveys, and feedback sessions.
- Clearly define and document requirements, roles, and responsibilities.
- Establish a governance structure with a steering committee to ensure alignment and decision-making.

2. Budget and Resource Constraints

Risks:

- Insufficient funding or resources to complete the project.
- Unexpected costs or resource shortages.

Impact:

- Incomplete or compromised features and functionalities.
- Project delays or failure.

Strategies:

- Develop a detailed budget plan with contingency provisions.
- Monitor expenditures and resource allocation closely.
- Seek additional funding sources or partnerships if necessary.
- Prioritize features and functionalities to focus on critical components first.

3. Legal and Regulatory Compliance

Risks:

- Non-compliance with local, national, or international laws and regulations.
- Issues related to intellectual property, trademarks, or copyrights.

Impact:

- Legal penalties and fines.
- Restrictions on operating in certain regions or markets.

Strategies:

- Conduct thorough legal reviews and consult with legal experts.
- Ensure all content and processes comply with relevant laws and regulations.
- Regularly update compliance policies and procedures.

4. Market and Competitive Risks

Risks:

- Changes in the market that affect the system's relevance or demand.
- Competitors launching similar or superior products.

Impact:

- Reduced user base and revenue.
- Difficulty in achieving market penetration and growth.

Strategies:

- Conduct market research to understand trends and competitor offerings.
- Develop a unique value proposition and continuously innovate.
- Gather and act on user feedback to improve and differentiate the system.

5. User Adoption and Engagement

Risks:

- Low user adoption due to lack of awareness or interest.
- Poor user engagement and retention.

Impact:

- Underutilization of the system and failure to achieve objectives.
- Negative perception and word-of-mouth.

Strategies:

- Implement a comprehensive marketing and outreach plan.
- Provide clear user guides, tutorials, and support.
- Engage users through regular updates, new features, and interactive elements.

6. Organizational and Operational Risks

Risks:

- Ineffective project management and coordination.
- High turnover of key personnel.

Impact:

- Project delays and reduced quality due to lack of continuity.
- Increased training and onboarding costs.

Strategies:

- Adopt robust project management methodologies (e.g., Agile, Scrum).
- Maintain comprehensive documentation and knowledge transfer processes.
- Foster a positive work environment to retain key staff.

7. Reputational Risks

Risks:

- Negative publicity due to issues like data breaches, unfair competition outcomes, or poor user experience.
- Miscommunication or mishandling of public relations.

Impact:

- Loss of trust and credibility.
- Decrease in user base and difficulty in attracting new users.

Strategies:

- Develop and implement a crisis management plan.
- Monitor public perception and address issues promptly and transparently.
- Engage in proactive communication and build strong relationships with the community.

8. Cultural and Ethical Risks

Risks:

- Cultural insensitivity in design or implementation.
- Ethical issues related to fairness and transparency in the competition.

Impact:

- Offending participants or stakeholders from diverse backgrounds.
- Legal challenges and loss of integrity.

Strategies:

- Ensure cultural competence and sensitivity in all aspects of the system.
- Establish and enforce ethical guidelines and fair practices.
- Regularly review and adapt policies to uphold integrity and inclusiveness.

Infrastructure

We will be using 'Trunk-based Development (No Branching) during the process of branching and merging in our team's repository.

Trunk-Based Development workflow

1. Develop directly on the trunk
    - Developers integrate small, frequent changes directly into the 'main' branch.
    - Each commit should be a self-contained unit of work that does not break the build or disrupt the system.
2. Continuous Integration (CI)
    - Automated tests are run on each commit to ensure new changes do not introduce issues.
    - The CI pipeline includes linting, unit tests, integration tests, and other relevant checks.

Steps for Everyday Development

1. Update Local Repository
    - Before starting any work, ensure the local repository is up to date with the main branch.
2. Make Changes

    - Develop the new feature or fix directly on the main branch in small increments.
    - Commit changes frequently with descriptive messages.

3. Push changes
    - Push commits to the remote main branch frequently to integrate with the work of others.
4. Automated Testing
    - The CI pipeline runs tests on each push to the main branch.
    - If tests pass, the changes are integrated; if tests fail, the developer must fix the issues immediately.

Handling Larger or Risky Changes

For larger or risky changes that cannot be done in small increments, a temporary short-lived branch might be used, though this is an exception rather than the norm.

1. Create a Temporary Branch
    - Create a branch from main for the larger change.
2. Develop on the feature branch
    - Make the necessary changes on the feature branch.
3. Frequent Integration
    - Rebase frequently to keep the branch up to date with main.
4. Merge Back to Trunk
    - Once the change is complete and tested locally, merge it back into main.

Code Review and Quality Assurance

1. Pull Requests for Exceptional Cases
    - When using a temporary branch, create a pull request (PR) to merge it back into main.
    - Ensure code reviews are part of the PR process to maintain code quality.
2. Code Reviews
    - Peer reviews focus on code quality, adherence to standards, and potential impacts.
    - At least one other team member should review the changes.
3. Approval and Merging
    - After passing code reviews and CI checks, approve and merge the PR.
    - Use merge commits or squash commits to maintain a clean history if needed.

Continuous Deployment (CD)

1. Automated Deployment
    - Deployments are automated from the main branch, ensuring the latest changes are always tested in a staging environment before going to production.

- Use feature flags to manage the deployment of incomplete features without impacting production.

2. Monitoring and Feedback
   - Continuously monitor the system for any issues after deployment.
   - Collect and act on feedback to iteratively improve the system.

Testing Tools and Automation

1. Version Control System

- Git: The primary tool for managing code changes, with a focus on maintaining a clean and stable main branch.

2. Continuous Integration (CI)

- Jenkins / Travis CI / CircleCI / GitHub Actions: Automate the build, test, and deployment processes. These tools trigger tests on every commit and provide feedback to developers.

3. Code Quality and Static Analysis

- ESLint / SonarQube: Tools for static code analysis to ensure adherence to coding standards and identify potential issues early.
- Prettier: Ensures consistent code formatting across the team.

4. Unit Testing

- Jest / Mocha / Chai: JavaScript testing frameworks for writing and running unit tests.
- JUnit / NUnit: Frameworks for unit testing in Java or .NET environments, respectively.

5. Integration Testing

- Postman / Newman: Tools for API testing to ensure different parts of the system work together correctly.
- Selenium / Cypress: Tools for end-to-end testing to automate browser interactions and verify the entire workflow of the application.

6. Automated Deployment

- Docker: Containerization for consistent environments across development, testing, and production.
- Kubernetes: Orchestration tool for managing containerized applications at scale.
- Terraform / Ansible: Infrastructure as Code (IaC) tools for automating the setup of infrastructure.

Competition Management System

7. Monitoring and Logging

- Prometheus / Grafana: Monitoring tools to track application performance and health.
- ELK Stack (Elasticsearch, Logstash, Kibana): Logging and analysis tools to collect and visualize logs for debugging and monitoring purposes.

Testing Processes

1. Automated Build and Test Pipeline

   A. Commit Stage
      - Every commit to the main branch triggers an automated build.
      - Static code analysis is performed to catch syntax errors and code style issues.
      - Unit tests are executed to verify individual components.

   B. Integration Stage
      - Upon passing unit tests, integration tests run to ensure different modules interact correctly.
      - API tests validate that endpoints respond as expected and handle edge cases.

   C. End-to-End (E2E) Testing Stage
      - E2E tests simulate user interactions to verify the system's overall functionality.
      - Tests cover

How to populate app with data

1. Continuous Data Import/Export

   a. API Endpoints

   - Develop API endpoints to allow external systems to integrate and exchange data with the competition management system. Ensure secure access with authentication and rate limiting.

2. Data Input from Users

   a. Forms and APIs

   - Create forms and APIs for users to input data directly into the system. Forms for competition creation, submission entries, and scoring need to be intuitive and user-friendly.

3. Initial Data Setup

   a. Database Schema Design

Competition Management System

- Design the database schema to support all required entities, such as Users, Competitions, Submissions, Scores, and Results.

Existing Solutions

Here is a list of some popular products and solutions along with an analysis of their similarities and differences compared to the proposed system:

1. Google Forms and Sheets

Similarities:

- Data Collection: Both can be used to collect participant information and competition entries.
- Basic Automation: Google Sheets can be used to automate basic tasks through scripts.

Differences:

- Complexity: Google Forms and Sheets are more suited for simple competitions and cannot handle complex workflows, automated scoring, or real-time updates.
- Integration: The proposed system would offer deeper integration with other tools and services specific to competition management, such as scoring systems and real-time leaderboards.
- Customization: The proposed system would provide more customization options specific to managing competitions, such as different competition types, automated judging, and participant communication.

2. Challonge

Similarities:

- Bracket Management: Both systems provide tools for creating and managing competition brackets.
- User-Friendly Interface: Both offer an interface for users to view competition progress and results.

Differences:

- Types of Competitions: Challonge is primarily focused on tournament-style competitions with elimination brackets. The proposed system would support a wider range of competition types, including points-based and multi-stage competitions.

Competition Management System

- Features: The proposed system might offer more comprehensive features such as automated scoring, detailed analytics, and integration with third-party services.
- Customization: Challonge has limited customization options compared to what a dedicated competition management system could provide.