Name – Aditya Sharma

Project Name – Data collection Backend Assignment (Atlan)

Project Assigned Date – 22/11/2023.

Project Completion Date – 26/11/2023

GITHUB LINK FOR CODEBASE - https://github.com/adityassharma-ss/data-collection-platform

Data Collection Platform - Design Specification

Overview

The Data Collection Platform is designed to provide a robust and flexible solution for managing forms, collecting responses, and integrating with external services. This document outlines the architectural design, key components, and features of the platform.

Architecture

The platform follows a three-tier architecture:

Presentation Tier (Frontend):

User interfaces for form creation, response submission, and integration management.

Utilizes HTML, CSS, and JavaScript for web-based interfaces.

Responsive design for optimal user experience across devices.

Application Tier (Backend):

Implemented using Node.js and Express for the server.

MongoDB as the primary database for storing forms and responses.

Modular controllers for handling form, response, and integration logic.

Integration with external services like Google Sheets and SMS gateways.

Data Tier (Database):

MongoDB for storing forms and responses.

Collections: forms, questions, responses, integrations.

Indexing for efficient querying.

Ensure eventual consistency for data integrity.

Components

1. Form Management

- Forms:

Stored in the forms collection.

Each form has metadata (e.g., title, description) and an array of questions.

Questions stored in the questions collection.

2. Response Management

- Responses:

Stored in the responses collection.

Each response is associated with a form and contains answer data.

Validation against business rules using the validation Service.

Integration with external services upon response submission.

3. Integration

- Google Sheets Integration:

Utilizes the google Sheets Service to update a designated Google Sheets spreadsheet.

Spreadsheet ID and API key configured in the .env file.

Maps response data to rows in the sheet.

- SMS Integration:

Utilizes the sms Service to send SMS receipts.

Integration with an SMS gateway service.

Configured in the .env file with service-specific credentials.

Eventual Consistency

Ensure eventual consistency across the platform.

Leverage MongoDB's replication features for data redundancy.

Implement retry mechanisms for failed integration attempts.

Failsafe and Recovery

Implement mechanisms for recovering from power, internet, or service outages.

Use logging to record errors and failures for debugging and analysis.

Scalability

Design to handle millions of responses across hundreds of forms.

Optimize database queries and indexing.

Implement load balancing and horizontal scaling if needed.

Security

Protect sensitive information (API keys, service credentials) using environment variables.

Implement user authentication and authorization for form and response management.

Regular security audits and updates.

Monitoring and Logging

Utilize logging for system health monitoring.

Set up alerts for critical issues or failures.

Monitor database performance and resource utilization.

Testing

Implement unit tests for controllers and services.

Integration tests for end-to-end testing of form creation, response submission, and integrations.

Conclusion

This design specification provides a detailed overview of the Data Collection Platform, covering its architecture, key components, features, and considerations for scalability, security, and monitoring. Follow the outlined design principles to ensure a robust and reliable platform for data collection and integration.

Project Development Methodology

1. Project Initiation:

- Define Objectives:

Clearly define the objectives of the Data Collection Platform.

Understand the primary goals, target users, and expected outcomes.

- Stakeholder Identification:

Identify and involve key stakeholders.

Understand their requirements and expectations.

- Feasibility Study:

Conduct a feasibility study to assess technical, economic, and operational viability.

Evaluate the potential challenges and risks.

2. Requirements Gathering:

- User Stories and Use Cases:

Create user stories and use cases to capture functional requirements.

Define the interactions between users and the system.

- Non-functional Requirements:

Identify and document non-functional requirements such as performance, security, and scalability.

- System Architecture:

Define the overall system architecture based on gathered requirements.

Choose technologies and frameworks.

3. Design Phase:

- Database Design:

Design the MongoDB database schema for storing forms, questions, responses, and integrations.

Define relationships between entities.

- Backend Design:

Design the backend architecture using Node.js and Express.

Define the structure of controllers, services, and routes.

- Frontend Design:

Design user interfaces for form creation, response submission, and integration management.

Implement responsive design for cross-device compatibility.

4. Implementation:

- Backend Development:

Implement the backend logic using Node.js and Express.

Connect to MongoDB for data storage.

- Frontend Development:

Develop the frontend interfaces using HTML, CSS, and JavaScript.

Implement user interactions and form validations.

- Integration Implementation:

Implement the integration services for Google Sheets and SMS.

Test and validate integration mechanisms.

5. Testing:

- Unit Testing:

Write unit tests for backend controllers and services.

Ensure individual components function correctly.

- Integration Testing:

Perform integration tests to validate interactions between components.

Test end-to-end scenarios for form creation, response submission, and integrations.

- User Acceptance Testing (UAT):

Engage stakeholders in UAT to ensure the platform meets their requirements.

Address feedback and make necessary adjustments.

6. Deployment:

- Environment Setup:

  Set up deployment environments (development, staging, production).

  Configure deployment scripts.

- Deployment:

  Deploy the platform to the production environment.

  Monitor for any issues during deployment.

7. Monitoring and Optimization:

- System Monitoring:

Set up monitoring tools to track system health, performance, and resource utilization.

Configure alerts for critical events.

- Optimization:

Optimize database queries, ensuring efficient data retrieval.

Address performance bottlenecks.

8. Documentation:

- Technical Documentation:

Create comprehensive technical documentation for the codebase.

Include API documentation and system architecture diagrams.

- User Documentation:

Develop user manuals and guides for using the platform.

Provide documentation for administrators and end-users.

9. Training:

- User Training:

Conduct training sessions for end-users on using the Data Collection Platform.

Provide support and answer any questions.

10. Maintenance and Support:

- Bug Fixes:

Address any post-deployment issues and bugs promptly.

- Continuous Improvement:

Gather feedback from users and stakeholders for future enhancements.

Plan and prioritize future releases based on feedback.

Summary of Implementation:

- Express Server Setup:

You set up an Express server as the backend framework to handle HTTP requests.

- MongoDB Integration:

Integrated MongoDB as the database to store forms, questions, and responses.

- Controller Setup:

Defined controllers (formController, responseController, integrationController) to handle different aspects of the application logic.

- Route Definitions:

Set up routes for handling form creation, response submission, and Google Sheets integration.

- Integration Services:

Implemented placeholder integration services (googleSheetsService, validationService, smsService) to manage Google Sheets integration, response validation, and SMS sending.

- Environment Variables

Utilized environment variables for sensitive information like MongoDB connection string, Google Sheets API key, and spreadsheet ID.

- Express Middleware:

Used express.json() middleware to parse JSON requests.

- Connection Status Logging:

Added console logs to indicate the status of MongoDB connection.

- NodeMon Usage:

Utilized NodeMon for automatic server restart during development.

Methodology Used:

- Modular-Code-Structure:
  Code is modularized into controllers and services, promoting maintainability and reusability.

- Environment-Variables:
  Sensitive information stored in environment variables for security.

- Express for RESTful API:
  Express used to create a RESTful API for form creation, response submission, and integration.

- Placeholder-Services:
  Placeholder services created for Google Sheets integration, response validation, and SMS sending, awaiting actual implementation.

- NodeMon for Development:
  NodeMon used for automatic server restart during development, improving the development workflow.

- Flexibility for Future Integrations:
  Placeholder services indicate a design approach ready for future integrations and extensions. Remember, this methodology prioritizes best practices for development, security, and maintainability, providing a solid foundation for building a robust and scalable data collection platform.

- **WHY-THIS-APPROACH?**

  The selected approach and methodology for this project were guided by a set of principles aimed at fostering a scalable, secure, and maintainable data collection platform. By adopting a modular structure with distinct controllers and services, the codebase becomes more manageable and adaptable to future enhancements without compromising existing functionalities. The use of environment variables ensures a heightened level of security, safeguarding sensitive information like API keys and database connections. Implementing a RESTful API with Express adheres to industry standards, ensuring compatibility and ease of integration with various clients and services. The incorporation of NodeMon streamlines the development workflow by automating server restarts upon code changes, boosting productivity during iterative development. Furthermore, the placeholder-based design not only allows for the systematic integration of sensitive operations, such as Google Sheets integration and SMS sending, but also lays

the groundwork for future extensions and integrations, offering a flexible and forward-looking solution. Overall, this methodology prioritizes best practices, security, and adaptability to create a robust foundation for the data collection-platform.

- **PROS-&-CONS-ANALYSIS:**

The selected approach and methodology for this project exhibit several notable advantages. The modular structure of the codebase ensures maintainability and extensibility, as each component operates independently, making it easier to introduce updates or incorporate new features without causing disruptions. By securely storing sensitive information, such as API keys and database connections, in environment variables, the codebase follows best practices for security, mitigating the risk of inadvertent exposure of critical credentials. The use of Express for building a RESTful API adheres to industry standards, ensuring compatibility with a diverse range of clients and services. The inclusion of NodeMon in the development workflow enhances productivity by automating server restarts upon code changes, streamlining the iterative development process. Additionally, the design's scalability and flexibility are noteworthy, with a modular and adaptable structure that accommodates future integrations seamlessly. The use of placeholder services for sensitive operations demonstrates a security-conscious approach, ensuring that security considerations are thoroughly addressed during actual implementation.

However, there are certain considerations to bear in mind. For developers unfamiliar with NodeMon, there may be a learning curve, potentially impacting initial productivity. The reliance on external services, such as Google Sheets and SMS gateways, introduces dependencies, making the platform susceptible to disruptions if these services encounter issues. Placeholder services, while strategic, imply additional work for full implementation, potentially delaying the complete functionality of the platform. Furthermore, the lack of detailed information on the SMS service's actual implementation introduces uncertainties when integrating this functionality. Care must be taken to manage environment variables securely, and potential configuration errors should be closely monitored across different environments. Overall, the chosen approach balances numerous advantages but requires careful consideration of potential challenges and dependencies during the development lifecycle.