This project exemplified vibe programming – a collaborative, iterative development approach marked by live interaction, real-time feedback, and organic problem-solving. Development progressed smoothly from concept to completion, with continuous refinement based on stakeholder input.

**Step-by-Step Design Journey using ChatGpt**

**ChatGpt Prompt:**

There is a standalone java application with dependency on java mail and Apache velocity that runs as a batch job, reads message details to be sent from database, using this data from table and velocity templates with static content and placeholders for key data like emailFrom, emailTo, Subject, caseNumber, and caseTitle, it sends an email to the user. I noticed that this application is part of a large monolithic application with a dozen more modules. I would like to replace this email system with JavaScript based solution. Please recommend a technology roadmap for this migration.

**ChatGpt Response:**

You have a few JavaScript options depending on environment and preference:

**Platform** Node.js + Express

**When to Choose** You need server-side batch processing

**Stack**  Node.js, Express.js, Nodemailer

**ChatGpt Prompt:**

I want to move email message database to Kafka

**ChatGpt Response:**

Great decision! Moving email messages from a relational database to **Kafka** decouples your system, improves scalability, and aligns with modern **event-driven architecture** principles.

Here’s a technology **roadmap** for migrating your **email message database to Kafka**, integrated with your new **JavaScript-based email service**.

**Old Flow:**

[Database Table] → [Java Batch Job] → [Velocity] → [JavaMail]

**New Flow with Kafka:**

[Producer System / Legacy App] → [Kafka Topic: email-messages] → [JavaScript Consumer Service] → [Template Engine] → [Nodemailer] → Email

**Summary of What to Add:**

| **Feature** | **New/Modified Files** | **Notes** |
| --- | --- | --- |
| Kafka client setup | /config/kafkaConfig.js | Broker URLs, group IDs |
| Kafka consumer logic | /kafka/consumer.js | Message listener, deserialization |
| Email service | /services/emailService.js | Template rendering + Nodemailer |
| App entry point | app.js | Initialize consumer, error handling |
| Logger | /utils/logger.js | Centralized logging |

**Step-by-Step Development Journey using Windsurf**

**Phase 1: Project Inception & Setup**

* **Initial Request:** The project was initiated with a requirement to integrate Nodemailer, Handlebars, Kafka Client, and a Consumer Service into a Node.js boilerplate.
* **Project Scaffolding:** A comprehensive Node.js project structure was established with a modular, maintainable architecture.
* **Dependency Management:** The package.json file was configured with all necessary dependencies, including KafkaJS, Nodemailer, Handlebars, and Express.
* **Environment Configuration:** A .env.example file was created to define email and Kafka configuration variables.

**Phase 2: Core Architecture Development**

* **Email Configuration:** A Nodemailer transporter was configured in src/config/email.js.
* **Kafka Configuration:** Producer and consumer settings were implemented in src/config/kafka.js.
* **Email Service:** The core email functionality was developed in src/services/emailService.js, featuring Handlebars template integration.
* **Kafka Services:** Kafka producer and consumer logic was implemented in src/kafka/producer.js and src/kafka/consumer.js, respectively.

**Phase 3: Template System & Examples**

* **Initial Templates:** Sample Handlebars templates for welcome and notification emails were created.
* **Example Scripts:** Test scripts and usage examples were provided to validate functionality.
* **Main Application:** The Express API was developed in src/index.js, including graceful shutdown handling.

**Phase 4: Template Conversion (Vibe Programming Moment)**

* **Discovery:** Existing Apache Velocity templates (.vm files) were identified.
* **Template Analysis:** case-notification.html.vm and case-notification.txt.vm were reviewed.
* **Conversion Process:** Velocity syntax (${var}) was converted to Handlebars syntax ({{var}}).
* **Enhancement:** Initially, advanced template features were introduced but were later simplified to meet stakeholder preferences.
* **Refinement:** Templates were finalized with a minimalistic structure through collaborative feedback.

**Phase 5: GitHub Integration & Debugging**

* **Repository Preparation:** The project was renamed to "NodeJsEmailer" and associated with a GitHub profile.
* **Git Setup:** A .gitignore file was created and repository details were added to package.json.
* **Bug Discovery:** A typo in the createTransporter method was identified during testing.
* **Live Debugging:** The issue was resolved in real time during the session.
* **Git Operations:** The repository was initialized, code committed, and prepared for GitHub push.

**Phase 6: Testing & Deployment**

* **Email Testing:** Test scripts were executed (authentication issues were noted, but overall code structure was validated).
* **Git Commit:** The code was committed with a custom message reflecting the collaborative development style.
* **GitHub Push:** A branch naming conflict (master vs. main) was encountered and addressed.

**Vibe Programming Characteristics Demonstrated**

* **Iterative Refinement:** Templates were enhanced and later simplified based on feedback.
* **Real-time Problem Solving:** Bugs and configuration issues were resolved immediately during testing.
* **Collaborative Decision Making:** Template structure and project architecture were guided by stakeholder input.
* **Organic Development Flow:** The solution evolved naturally from a simple boilerplate to a specialized case notification system.
* **Live Debugging:** Issues were tackled as they arose, minimizing downtime.
* **Flexible Architecture:** The system was designed to accommodate future extensions with ease.

**Final Deliverables**  
**Complete NodeJsEmailer Project with:**

* Nodemailer + Handlebars + Kafka integration
* Missouri Judiciary case notification templates (converted from Velocity)
* REST API endpoints for email sending and queuing
* Comprehensive documentation and examples
* Production-ready configuration and error handling
* Git repository prepared for GitHub (final push step pending)

This project showcased a successful application of vibe programming – a responsive and collaborative development style that produced a polished, extensible, and production-ready email notification service tailored for case-based workflows.