CampusSync

Web Architecture Documentation

Version 1.0.0

Generated: 11/15/2025, 9:33:51 AM

System Overview

Technology Stack

CampusSync is a full-stack PERN (PostgreSQL, Express, React, Node.js) web application designed to help students manage academic overhead through automated email alert detection and AI-generated flashcards.

Frontend:

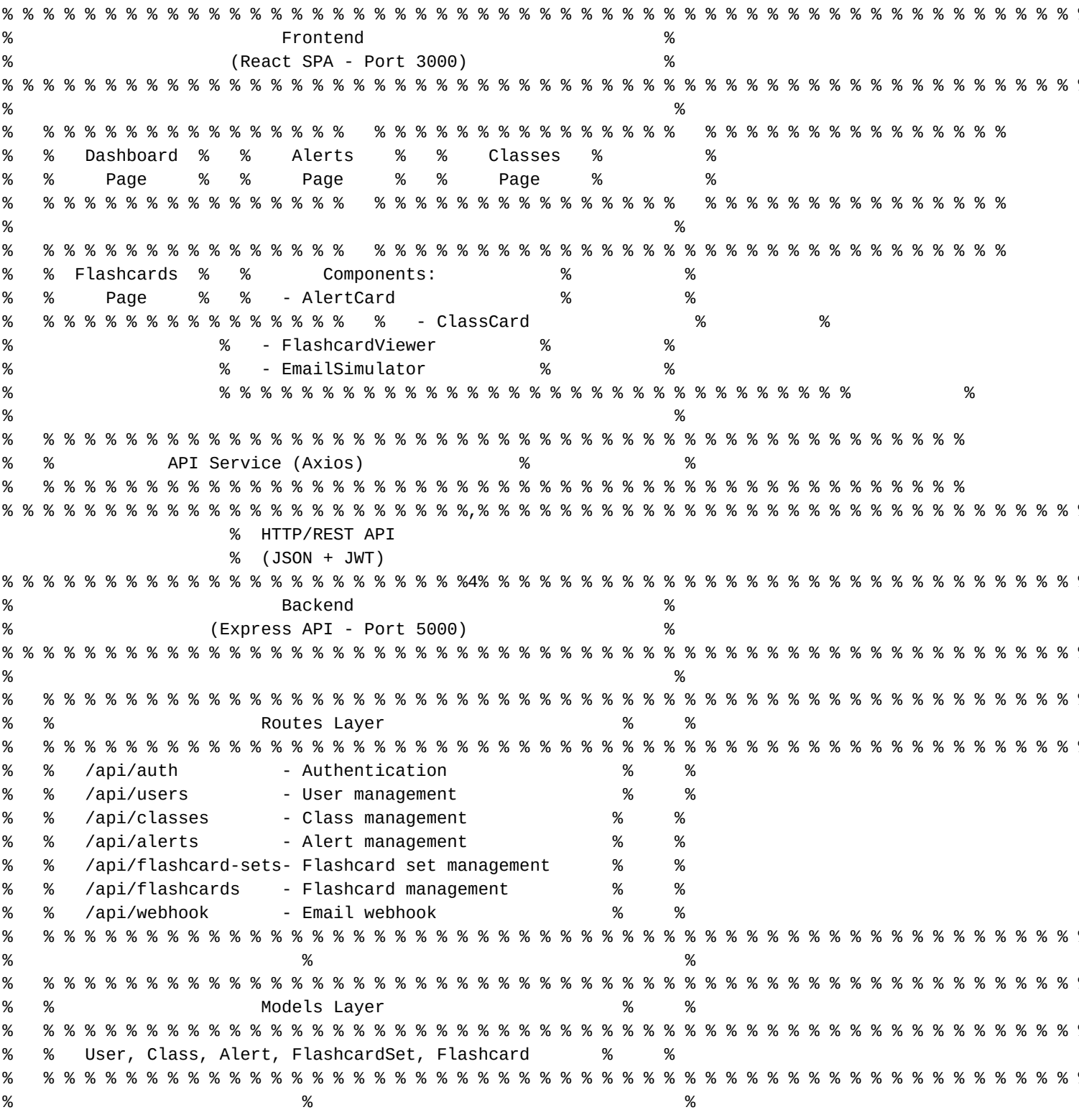
- React 18.2.0 - UI framework
- React Router DOM 6.20.0 - Client-side routing
- Axios 1.6.2 - HTTP client for API calls
- CSS3 - Modern styling

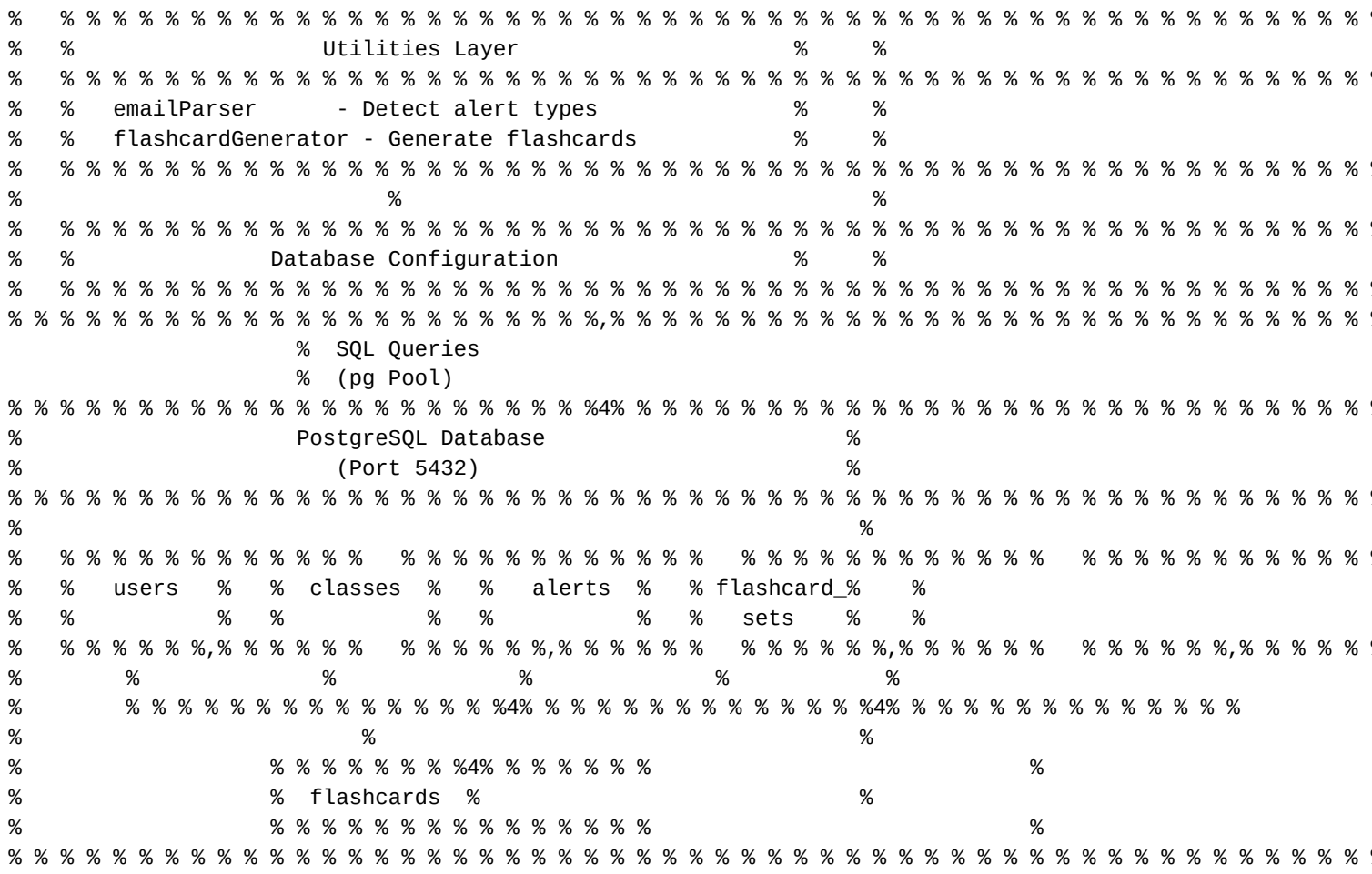
Backend:

- Node.js - JavaScript runtime
- Express 4.18.2 - Web application framework
- PostgreSQL - Relational database
- pg 8.11.3 - PostgreSQL client
- JWT - Authentication tokens
- bcrypt - Password hashing

System Architecture

High-Level Architecture




Utilities Layer
emailParser - Detect alert types
flashcardGenerator - Generate flashcards
Database Configuration
SQL Queries
(pg Pool)
PostgreSQL Database
(Port 5432)
users
classes
alerts
flashcard_sets
flashcards

Data Flow Diagrams

1. Authentication Flow

```
User Action (Login/Signup)
    %
    % % > Frontend: Login/Signup Component
    %        %
    %        % % > POST /api/auth/login
    %        %    or POST /api/auth/signup
    %        %
    % % > Backend: auth.js Route
    %        %
    %        % % > User.findByEmail() / User.createWithPassword()
    %        %
    %        % % > Password verification (bcrypt)
    %        %
    %        % % > Generate JWT token
    %        %
    %        % % > Response with user + token
    %
    % % > Frontend: Store token in localStorage
            %
            % % > Redirect to Dashboard
```

2. Email Alert Detection Flow

```
Email Webhook / User Input (Email Simulator)
    %
    % % > POST /api/webhook/email
    %    or POST /api/alerts/process-email
    %
    % % > Backend: webhook.js / alerts.js Route
    %        %
    %        % % > emailParser.processEmail()
    %        %        %
    %        %        % % > detectAlertType() - Pattern matching
    %        %        % % > extractClassInfo() - Data extraction
    %        %
    %        % % > Alert.create() - Save to database
    %        %
    %        % % > Response with alert object
    %
    % % > Frontend: Update alerts list
            %
            % % > Display new alert in AlertCard
```

3. Flashcard Generation Flow

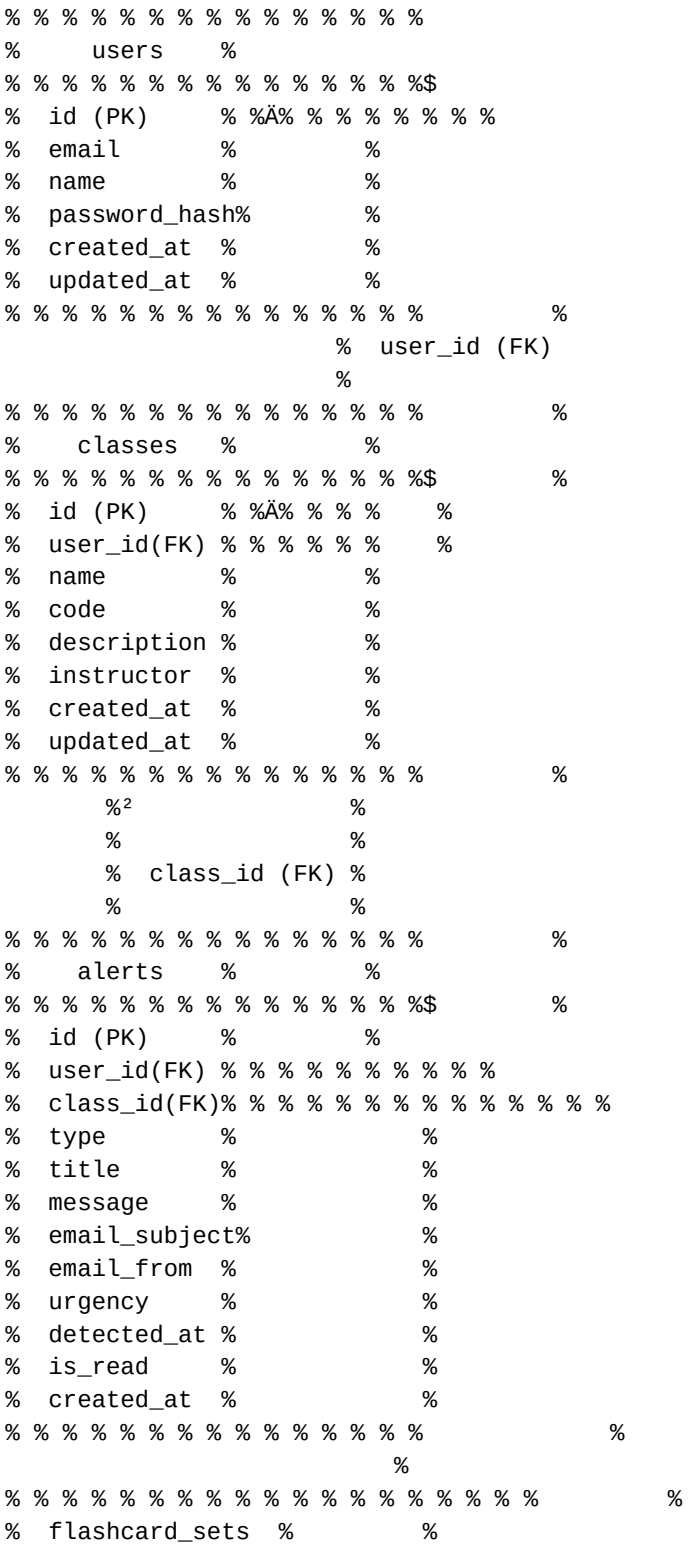
```
User Input (Create Flashcard Set)
    %
    % % > Frontend: FlashcardSetForm Component
    %         %
    %         % % > POST /api/flashcard-sets/generate
    %         %
    % % > Backend: flashcardSets.js Route
    %         %
    %         % % > FlashcardSet.create() - Create set
    %         %
    %         % % > flashcardGenerator.generateFlashcardsByTopic()
    %         %         %
    %         %         % % > Generate question-answer pairs
    %         %
    %         % % > For each flashcard:
    %         %         % % > Flashcard.create() - Save to database
    %         %
    %         % % > Response with set and flashcards
    %
    % % > Frontend: Update flashcard sets list
             %
             % % > Display new set in FlashcardSetCard
```

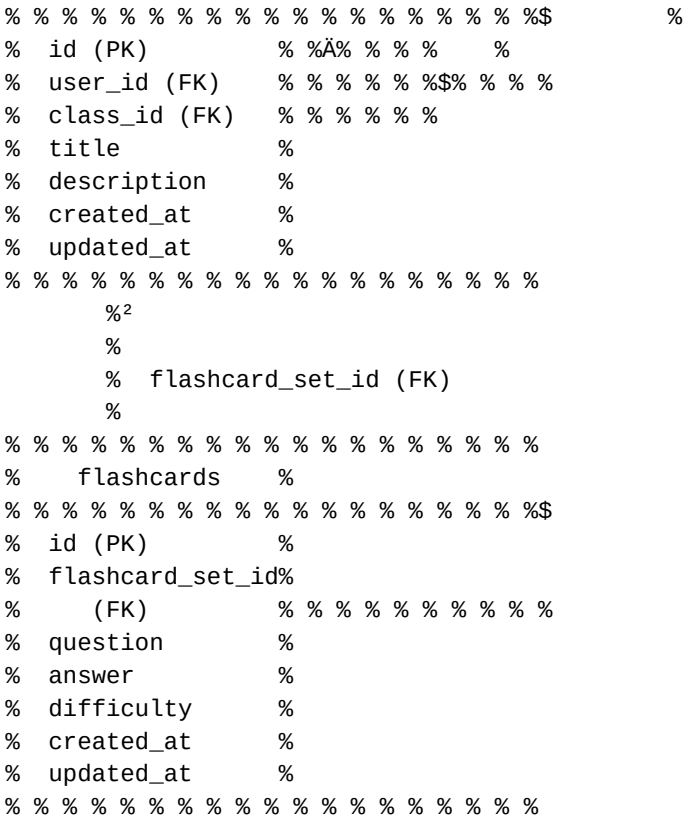
4. Study Session Flow

```
User clicks "Study" on a flashcard set
    %
    % % > Frontend: FlashcardViewer Component
    %         %
    %         % % > GET /api/flashcards/set/:setId
    %         %
    % % > Backend: flashcards.js Route
    %         %
    %         % % > Flashcard.findBySetId() - Query database
    %         %
    %         % % > Response with array of flashcards
    %
    % % > Frontend: Interactive study mode
             %
             % % > Display current flashcard
             % % > Flip animation (question !” answer)
             % % > Navigation (previous/next)
             % % > Progress tracking
```

Database Schema

Entity Relationship Diagram




id (PK)
user_id (FK)
class_id (FK)
title
description
created_at
updated_at
flashcard_set_id (FK)
flashcards
id (PK)
flashcard_set_id
(FK)
question
answer
difficulty
created_at
updated_at

API Endpoints

RESTful API Structure

```
/api
% % %  /auth
%    % % %  POST   /signup        - Create user account
%    % % %  POST   /login         - Authenticate user
%    % % %  GET    /me            - Get current user (protected)
%
% % %  /users
%    % % %  GET    /              - List all users
%    % % %  GET    /:id           - Get user by ID
%    % % %  POST   /              - Create user
%    % % %  PUT    /:id           - Update user
%    % % %  DELETE /:id           - Delete user
%
% % %  /classes
%    % % %  GET    /user/:userId  - Get classes for user
%    % % %  GET    /:id           - Get class by ID
%    % % %  POST   /              - Create class
%    % % %  PUT    /:id           - Update class
%    % % %  DELETE /:id           - Delete class
%
% % %  /alerts
%    % % %  GET    /user/:userId     - Get alerts for user
%    % % %  GET    /:id              - Get alert by ID
%    % % %  POST   /                  - Create alert
%    % % %  POST   /process-email     - Process email & create alert
%    % % %  PATCH  /:id/read         - Mark alert as read
%    % % %  DELETE /:id              - Delete alert
%
% % %  /flashcard-sets
%    % % %  GET    /user/:userId  - Get sets for user
%    % % %  GET    /:id           - Get set by ID
%    % % %  POST   /              - Create set
%    % % %  POST   /generate      - Create set with AI flashcards
%    % % %  PUT    /:id           - Update set
%    % % %  DELETE /:id           - Delete set
%
% % %  /flashcards
%    % % %  GET    /set/:setId    - Get flashcards for set
%    % % %  GET    /:id           - Get flashcard by ID
%    % % %  POST   /              - Create flashcard
%    % % %  PUT    /:id           - Update flashcard
%    % % %  DELETE /:id           - Delete flashcard
%
% % %  /webhook
    % % %  POST   /email         - Email webhook endpoint
```

Frontend Component Hierarchy

React Component Tree

```
App
% % %  Router
    % % %  Navbar
    %    % % %  Logo
    %    % % %  Navigation Menu
    %    % % %  User Info
    %
    % % %  Dashboard (/)
    %    % % %  Stats Cards
    %    %    % % %  Alerts Stat
    %    %    % % %  Classes Stat
    %    %    % % %  Flashcards Stat
    %    % % %  Recent Alerts List
    %
    % % %  Alerts (/alerts)
    %    % % %  Alerts Toolbar
    %    %    % % %  Filter Buttons
    %    %    % % %  Email Simulator Toggle
    %    % % %  EmailSimulator
    %    %    % % %  Template Buttons
    %    %    % % %  Email Form
    %    % % %  Alerts Grid
    %        % % %  AlertCard (multiple)
    %            % % %  Alert Header
    %            % % %  Alert Content
    %            % % %  Alert Actions
    %
    % % %  Classes (/classes)
    %    % % %  Page Header
    %    % % %  ClassForm (conditional)
    %    % % %  Classes Grid
    %        % % %  ClassCard (multiple)
    %            % % %  Class Info
    %            % % %  Actions
    %
    % % %  Flashcards (/flashcards)
    %    % % %  FlashcardSetForm (conditional)
    %    %    % % %  Basic Info Fields
    %    %    % % %  AI Generation Toggle
    %    %    % % %  Topic Input
    %    % % %  Flashcard Sets Grid
    %    %    % % %  FlashcardSetCard (multiple)
    %    %        % % %  Set Info
    %    %        % % %  Actions
    %    % % %  FlashcardViewer (conditional)
    %        % % %  Viewer Header
    %        % % %  Progress Bar
```

```
%        % % %  Flashcard Display
%        %    % % %  Question Side
%        %    % % %  Answer Side
%        % % %  Navigation Controls
%
% % %  Login (/login)
%    % % %  Login Form
%
% % %  Signup (/signup)
    % % %  Signup Form
```

Key Features & Implementation

1. Authentication System

• JWT-based authentication
• Password hashing with bcrypt
• Protected routes with token verification
• Token stored in localStorage
• Auto-logout on token expiration

2. Email Alert Detection

Algorithm:

1. Receive email content (from, subject, body)
2. Combine subject and body into searchable content
3. Convert to lowercase for case-insensitive matching
4. Iterate through predefined keyword patterns:
   - cancellation: "class cancelled", "no class", etc.
   - exam_change: "exam moved", "test rescheduled", etc.
   - extra_credit: "extra credit", "bonus points", etc.
   - assignment: "assignment due", "homework due", etc.
   - schedule_change: "room change", "time change", etc.
5. Return first matching type
6. Generate appropriate title and message
7. Save to database with urgency level

3. AI Flashcard Generation

Current Implementation (Template-based):

1. Accept topic and count from user
2. Use template-based generation:
   - "Define [topic]"
   - "What are the main characteristics of [topic]?"
   - "How does [topic] relate to other concepts?"
   - "Why is [topic] important?"
   - "Give an example of [topic]"
3. Assign difficulty levels based on question complexity
4. Save to database

Security & Deployment

Security Features

- SQL Injection Prevention: Parameterized queries using pg library
- CORS Configuration: Configured to allow frontend origin
- Input Validation: Required field validation in routes
- Password Security: bcrypt hashing with salt rounds
- JWT Tokens: Secure token-based authentication
- Error Handling: Proper error messages without exposing internals

Development Environment

```
Localhost:3000 (React Dev Server)
    %
    % % > Localhost:5000 (Express API)
    %        %
    %        % % > Localhost:5432 (PostgreSQL)
```

Production Recommendations

- HTTPS: Encrypt data in transit with SSL/TLS certificates
- Environment Variables: Store secrets in .env (not committed)
- Rate Limiting: Prevent API abuse and DDoS attacks
- Database Backups: Regular automated backups
- Monitoring: Application performance monitoring (APM)
- Logging: Centralized logging system
- CDN: Serve static assets from CDN
- Load Balancing: Multiple API server instances

Architecture Summary

System Characteristics

CampusSync's architecture is designed to be:

• Modular: Clear separation of concerns
• Scalable: Can handle growth in users and data
• Maintainable: Well-organized code structure
• Extensible: Easy to add new features
• Secure: Security best practices implemented
• Performant: Efficient queries and rendering

Technology Stack Summary

Frontend: React + React Router + Axios
Backend: Node.js + Express + PostgreSQL
Authentication: JWT + bcrypt
Database: PostgreSQL with connection pooling

--- End of Document ---
Generated: 11/15/2025, 9:33:51 AM