Backend Development

Sampath Kumar

4SF22CS175

CS - 5B

tce.

Introduction

Objective: Simplifies utility bill payments for Mangalore citizens by providing a streamlined platform for electricity, water, and gas bill management.

Features: Processes payment requests, generates printable PDF invoices, and handles urgent cases (like overdue payments) with priority.

Technology Stack: Built using Node.js, Express, and file handling, incorporating data structures like stacks and queues to manage transactions efficiently.

Problem Statement

"The citizens of Mangalore face frequent issues with **timely utility bill payments** due to inefficiencies in the current systems, leading to **delays and inconvenience.** There is a pressing need for a solution that can manage **regular and urgent requests,** such as **overdue payments** and disconnections, with high priority to prevent service disruptions. This project addresses these challenges by creating an organized utility bill payment system that **processes payments efficiently**, handles urgent requests promptly, and **maintains comprehensive records** for services like electricity, water, and gas, ultimately improving accessibility and transparency."

Solution

System requirements for the project

Node.js: Runtime environment for executing JavaScript on the server. (Recommended version: 14.x or higher)

Express.js: Web application framework for building APIs and handling server routes.

File System (fs): Node.js core module for handling file operations, used for reading and writing transaction records.

PDFKit (or an alternative PDF library): Generates PDF invoices for utility bills.

Nodemon (optional for development): Automatically restarts the server upon file changes, useful during development.

Postman (or an equivalent API testing tool): Tests API endpoints and verifies data flow.

Git: Version control system to manage source code and collaborate on project updates.

Visual Studio Code (or any preferred code editor): Recommended for writing, testing, and debugging the code.

Solution

Flowchart of the project

User Inputs and API Request: Users enter bill details (name, utility type, amount, and due date) through a form, which sends a request to the server to process payment or generate an invoice.

Request Processing and Queue Management: Server routes the request, managing it through a queue. Urgent requests (e.g., overdue bills) are prioritized.

Transaction Logging and Invoice Generation: The server logs each transaction to a JSON file for record-keeping and generates a PDF invoice for the user.

Response and File Download: The server sends back a downloadable invoice file to the user, confirming payment and storing transaction details for audit purposes.

Solution

Backend

API Development: Developed APIs using Express.js to handle core functionalities like bill payments, invoice generation, and transaction logging.

Routing and Middleware: Defined routes for different endpoints and used middleware for request validation, error handling, and logging.

PDF Generation: Integrated a custom service for dynamic PDF invoice creation, ensuring invoices are downloadable after submission.

Error Handling: Implemented robust error handling to manage exceptions and ensure smooth backend performance.

Solution

Array use case demonstration

Data Storage: Used arrays to temporarily store transaction data for quick access and manipulation during runtime.

Filtering and Sorting: Utilized array methods like filter() and sort() to organize and retrieve data based on user preferences or date.

Dynamic JSON Conversion: Converted array data to JSON format for writing to files and ensuring persistence.

Summary Reports: Demonstrated how array traversal and aggregation methods are used to generate transaction summaries.

Solution

Stack and Queue in solving problem

Stack Implementation: Used to keep track of user interactions or undo operations, allowing efficient backtracking in multi-step forms.

Queue for Request Management: Implemented a queue system for processing multiple payment requests in a sequential (FIFO) order.

Priority Queue: Used in scenarios where certain tasks (e.g., overdue payments) need to be prioritized over regular tasks.

Efficient Task Scheduling: Demonstrated better task flow management during high system load by ensuring no request is missed.

Solution

Demonstrate Priority factor in problem statement

Dynamic Priority Assignment: Assigned higher priority to overdue or high-value payments for faster processing.

Priority Queue Usage: Utilized a priority queue to handle requests based on urgency rather than arrival time.

Real-World Use Case: Demonstrated in cases where system needs to process emergency payments before due dates.

Optimized User Experience: Ensures critical operations are handled efficiently, reducing late payment risks.

Solution

Demonstrate how are you using File handling in problem statement

Data Persistence: Stored all transaction records in a transactions.json file to maintain a historical log.

Reading and Writing: Used fs module for reading existing transactions and appending new ones securely.

Data Backup: Ensured records are not lost by maintaining file-based storage that persists beyond runtime.

Audit and Reporting: Enabled generation of periodic reports by reading and summarizing data from the transaction file.

Conclusion

Efficient Invoice Generation: The system dynamically generates and provides downloadable PDF invoices for each user transaction, ensuring a seamless billing process.

Real-time Data Management: Implements file handling to store and retrieve transaction records, ensuring that all payment details are securely logged and accessible for auditing purposes.

Priority Handling for Payments: Utilizes a priority-based system to ensure urgent bills are processed first, improving user satisfaction and system reliability.

Scalable and Robust Backend: Incorporates arrays, stacks, and queues to efficiently handle multiple concurrent transactions and operations, meeting the system's performance requirements.

References

1. Node.js Documentation
2. Express.js Guide
3. File System (fs) Module in Node.js
4. PDFKit Documentation for PDF Generation
5. Git Documentation for Version Control
6. JSON File Handling in Node.js
7. Queue and Stack Concepts in JavaScript

Thank You

tce.