Billify App

SaaS Billing App Design & Implementation Using Cloudflare Workers

Project Overview:

This project is a serverless billing app for a SaaS platform, developed with **Cloudflare Workers** and **TypeScript** to manage subscriptions, recurring billing, and payment processing.

Why Cloudflare Workers?

Cloudflare Workers provide a scalable, cost-effective serverless environment ideal for real-time processing. With Workers, the app benefits from:

- **Low Latency**: Cloudflare's extensive edge network ensures fast responses globally.
- **Scalability**: Workers scale effortlessly with traffic demands, making them perfect for handling unpredictable usage.
- **Cost Efficiency**: A pay-as-you-go model reduces overhead for smaller workloads.

Core Features:

1. **Subscription Management**: Create/manage subscription plans, assign plans to customers, and track subscription status.
2. **Billing Engine**: Automatically generate invoices based on customer plans, with proration for mid-cycle changes.
3. **Payment Processing**: Record and update payment statuses and handle retry logic for failed payments.
4. **Notifications**: Send email alerts for invoices and payment events (e.g., SendGrid/Mailgun).

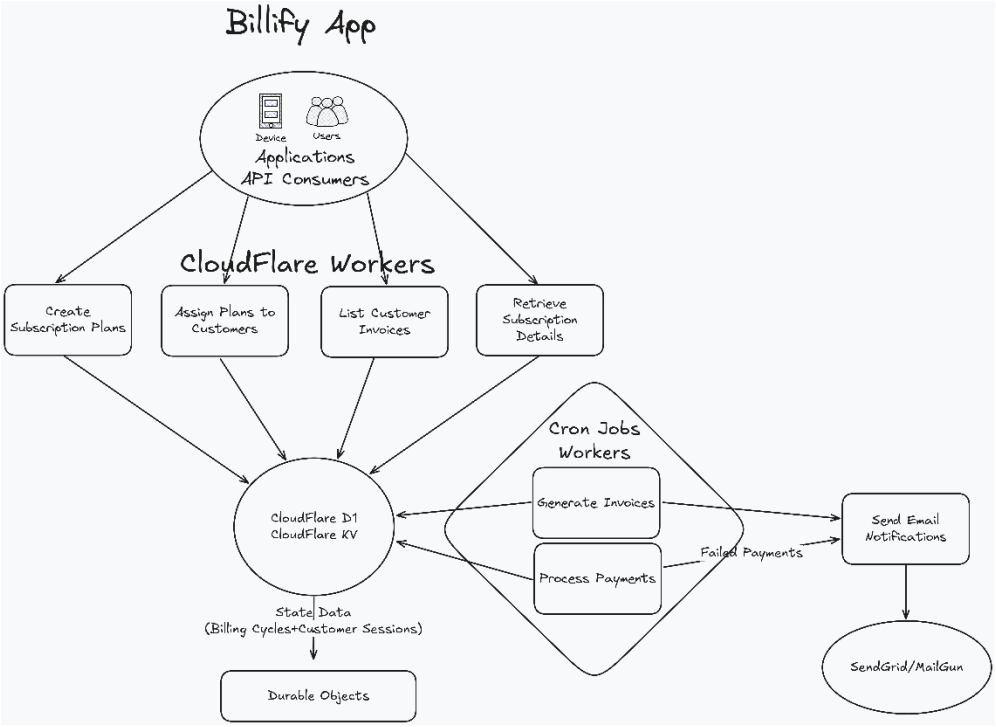
Additional Features:

- **Invoice Generation Function**: Generates invoices on-demand or via scheduled events.
- **Data Storage**: Uses Cloudflare KV and Durable Objects to store subscription, invoice, and payment data.
- **API Endpoints**: Handles creating subscriptions, invoices, payments, listing customer invoices, etc.

Architecture Diagram

The Architecture Diagram visualizes the flow of data and interactions across various components in the billing application. Key components include:

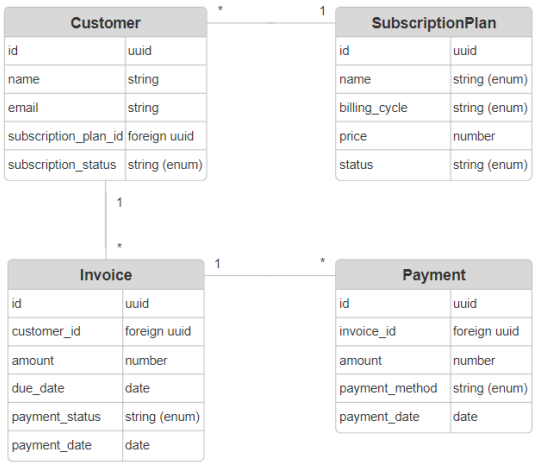
- **Cloudflare Workers**: Primary serverless backend handling subscription management, billing logic, and API endpoints.
- **Durable Objects**: Manages stateful data for customer sessions, billing cycles, and subscription statuses.
- **Cloudflare KV**: Stores customer data, invoices, payments, and subscription details for quick retrieval.
- **Event Queue (Future Enhancement)**: Connects components for async event handling, such as sending notification requests to a dedicated notifications worker.
- **Email Service (e.g., SendGrid or Mailgun)**: Handles email notifications for invoice generation, payment status, and failed payment alerts.



ERD (Entity-Relationship Diagram)

The ERD outlines the relationships between key entities in the billing system, such as Customer, SubscriptionPlan, Invoice, and Payment.

Billify Entity Relationship Diagram



Relationships:

1. A Customer can have one SubscriptionPlan,
   A SubscriptionPlan can have multiple Customers

2. A Customer can have multiple Invoices,
   An Invoice can have one Customer

3. An Invoice can have multiple Payments,
   A Payment can have one Invoice.

Assumptions:

1. For each invoice there could be multiple payments, if a payment fails, a new payment is processed for the same invoice previously issued.

What Could Be Better?

- **Data Storage Optimization**: Using Cloudflare KV has limitations, especially for complex queries or relational data handling.
- **Improved Testing**: More comprehensive test coverage, especially for edge cases, could improve reliability.
- **Error Handling and Response Consistency**: More work on standardized error handling and responses would improve API usability and debugging.

If I Had More Time (Steps for Improvements):

1. **Switch to D1 for SQL Support**: Cloudflare's D1 offers a serverless SQL solution, providing better data management and relational queries than KV.
2. **Enhance Durable Objects Usage**: Deepen the implementation of Durable Objects for managing customer sessions and billing cycles, enhancing statefulness.
3. **Implement Event Queues**: Use Cloudflare Queues to handle event-driven communication between workers (e.g., notifying a dedicated notifications worker).
4. **Adopt TDD**: Prioritize Test-Driven Development (TDD) to ensure stable and reliable features from the start.
5. **Integrate a Payment Gateway**: Use Stripe or Tap for secure, reliable payment processing and compliance.
6. **Standardize Response Schemas:** Build consistent schemas for HttpResponses and ErrorResponse, including clear documentation for each.

Resources:

- **Cloudflare Developers YouTube**: YouTube Channel
- **Cloudflare Documentation**: Documentation
- **Hono Framework**: Docs
- **Hono GitHub Repository**: GitHub
- **Validation with Zod**: Documentation
- **Mailgun for Email Integration**: Documentation
- **Trello (for Project Task Management):** Trello
- **Excalidraw (for Diagramming and Visualization):** Excalidraw
- **SmartDraw (for Workflow and Diagramming):** SmartDraw
- **Postman (for API Testing):** Postman