

Invitewala Platform - Complete Technical Documentation

Wedding Card Personalization & WhatsApp Distribution Platform

Version 2.0 - Enhanced Multi-User Event Card Management System

Table of Contents

1. [Overview](#)
 2. [System Architecture](#)
 3. [Technology Stack](#)
 4. [User Roles & Access Control](#)
 5. [Project Structure](#)
 6. [Core Features & Modules](#)
 7. [Backend Components \(Detailed\)](#)
 8. [Frontend Components \(Detailed\)](#)
 9. [PDF Personalizer Engine](#)
 10. [Database Schema](#)
 11. [API Reference](#)
 12. [Authentication & Authorization](#)
 13. [Design Studio Workflow](#)
 14. [Customer Management Module](#)
 15. [Dashboard & Analytics](#)
 16. [WhatsApp Integration System](#)
 17. [PDF Management & Delivery](#)
 18. [Notification System](#)
 19. [Settings & Configuration](#)
 20. [Workflows & Business Logic](#)
 21. [File Storage System](#)
 22. [Deployment Guide](#)
 23. [Troubleshooting](#)
-

Overview

Invitewala is an enterprise-grade, multi-user platform that enables printing businesses and event organizers to create, customize, and distribute personalized invitation cards at scale.

Core Capabilities

1. **Multi-User Role Management:** Super Admin, Admin, and Customer hierarchies with role-based access control
2. **Visual Design Studio:** Upload PDF templates and define customizable text zones using visual editor
3. **Bulk Guest Management:** Import guest lists from CSV files with validation
4. **Automated Personalization:** Generate unique invitation cards for each guest
5. **WhatsApp Distribution:** Send cards via WhatsApp Web or Cloud API with delivery tracking
6. **Demo Approval Workflow:** Create demo cards for customer approval before bulk generation
7. **Real-time Analytics:** Track sent cards, delivery status, and RSVP responses
8. **Event Management:** Manage multiple events per customer with separate guest lists
9. **Zone Conflict Prevention:** Intelligent zone management to avoid overlapping content
10. **PDF Lifecycle Management:** Auto-cleanup after event completion

Target Users

- **Wedding invitation printing businesses:** Manage multiple customers and events
- **Event planning companies:** Corporate events, birthday parties, anniversaries
- **Marketing agencies:** WhatsApp-based invitation campaigns
- **Individual event organizers:** Self-service card creation and distribution

Key Differentiators

- **Multi-tenancy support:** Super Admin can manage multiple Admin accounts
- **Customer isolation:** Each Admin sees only their assigned customers
- **Demo-before-production workflow:** Reduce errors with approval gates
- **Duplicate prevention:** Never send the same PDF twice to a guest
- **Zone-based customization:** Advanced masking and overlay options
- **Delivery tracking:** Real-time WhatsApp delivery status per guest

System Architecture

High-Level Architecture

Client Layer (Browser)

Login & Auth Dashboard & Stats Design Studio Customer Tab Settings & Admin

HTTPS/REST API

FastAPI Backend (Port 5234)

API Layer (Routes)

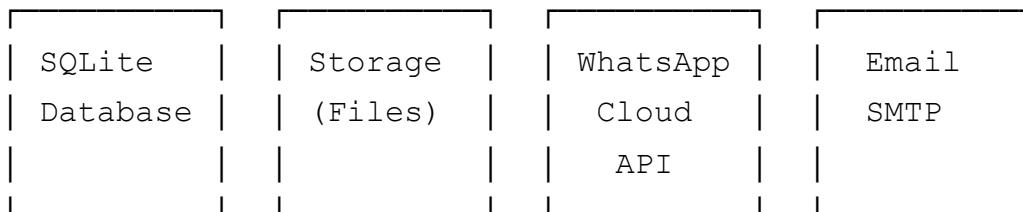
auth user customer design whatsap
mgmt mgmt mgmt studio api

Business Logic Layer

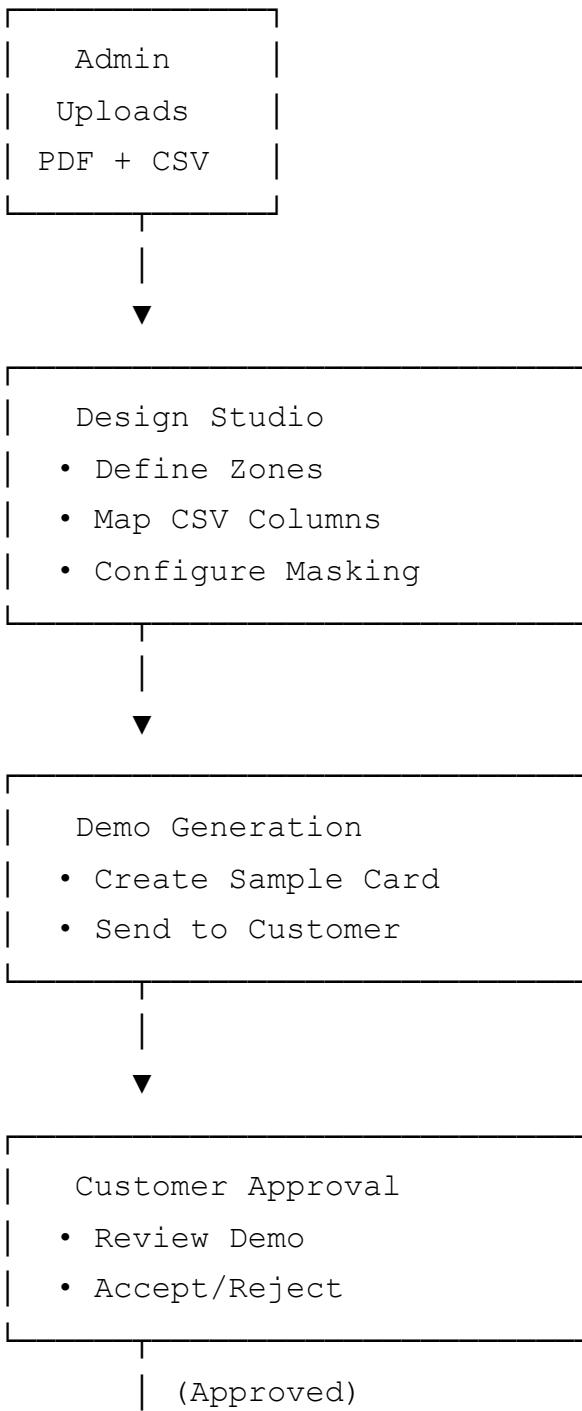
Zone Conflict Prevention Demo Approval Workflow Delivery Tracking System

PDF Personalization Engine

Template Loader Zone Masking Text Overlay Batch Generator



Data Flow Architecture



Batch Generation

- Process All CSV Rows
- Generate Individual PDFs
- Store with Guest Info

Customer Tab

- Admin Selects Guests
- Views Generated PDFs
- Chooses Send Options

WhatsApp Delivery

- Send via API
- Track Delivery
- Update Status
- Prevent Duplicates

Dashboard Analytics

- Update Statistics
- Track Success Rate
- Show Delivery Status

Technology Stack

Backend Technologies

Component	Technology	Version	Purpose
-----------	------------	---------	---------

Component	Technology	Version	Purpose
Framework	FastAPI	0.100+	High-performance async API with automatic OpenAPI docs
Database	SQLite	3.35+	Lightweight embedded relational database
PDF Processing	PyMuPDF (fitz)	1.23+	PDF manipulation, text extraction, and rendering
Image Processing	Pillow (PIL)	10.0+	Image generation, manipulation, and preview creation
Authentication	PyJWT	2.8+	JSON Web Token creation and verification
Password Security	bcrypt	4.0+	Secure password hashing with salt
Server	Uvicorn	0.24+	Lightning-fast ASGI server for async applications
HTTP Client	httpx	0.25+	Async HTTP client for WhatsApp API calls
Data Validation	Pydantic	2.0+	Data validation using Python type annotations
CSV Processing	Python csv module	Built-in	CSV parsing and validation
Email	smtplib/email	Built-in	Email notifications and password reset

Frontend Technologies

Component	Technology	Version	Purpose
Framework	React	18.2+	Component-based UI library with hooks
Build Tool	Vite	5.0+	Next-generation frontend tooling with HMR
Language	JavaScript/JSX	ES2022	Modern JavaScript with React syntax
Styling	TailwindCSS	3.3+	Utility-first CSS framework
Routing	React Router	6.18+	Declarative routing for single-page apps
State Management	React Context	18.2+	Global state without external dependencies
HTTP Client	Fetch API	Built-in	Native browser API for HTTP requests
Form Handling	React useState	Built-in	Controlled form components
Icons	Lucide React	0.290+	Beautiful & consistent icon set
Charts	Chart.js	4.4+	Flexible charting library for analytics

Infrastructure & DevOps

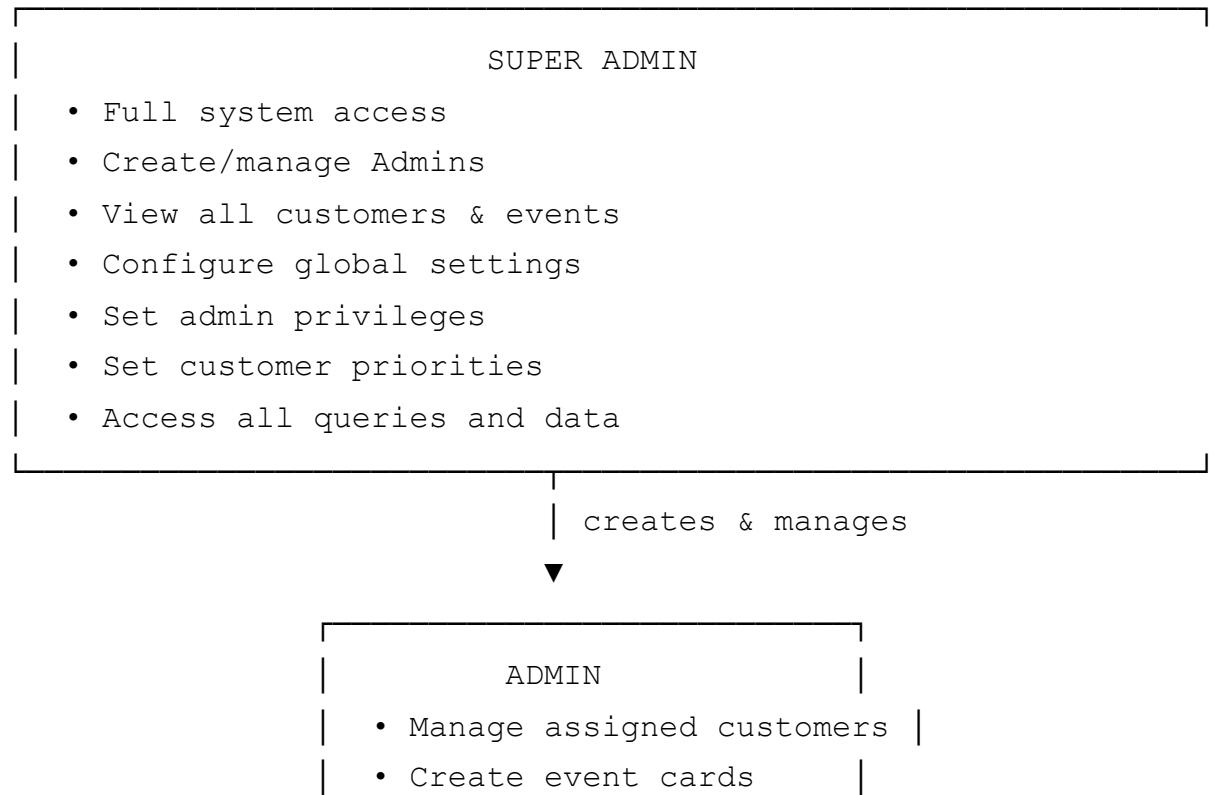
Component	Technology	Purpose
Version Control	Git	Source code management
Process Manager	systemd/PM2	Production process management
Reverse Proxy	Nginx(optional)	Load balancing & SSL termination
Containerization	Docker(optional)	Consistent deployment environments
SSL/TLS	Let's Encrypt	Free SSL certificates
Monitoring	System logs	Application and error monitoring

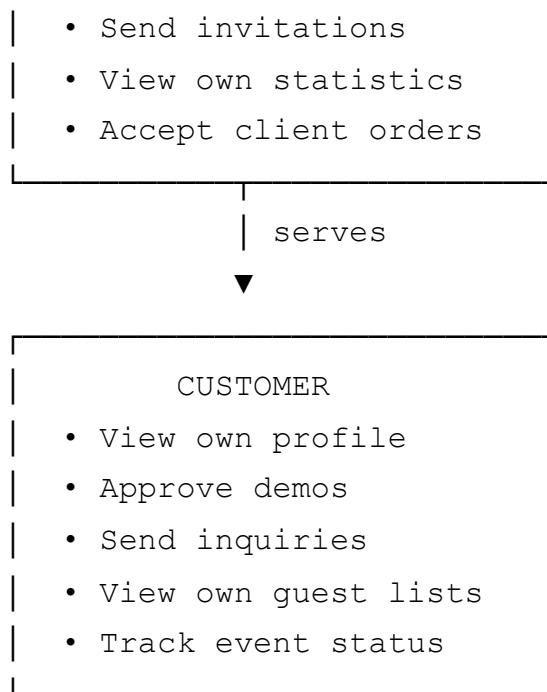
External Services

Service	Purpose	Provider Options
WhatsApp API	Message delivery	WhatsApp Cloud API / Twilio / custom
SMTP Server	Email notifications	Gmail / SendGrid / Mailgun
Storage	File storage (optional)	S3 / Local filesystem

User Roles & Access Control

Role Hierarchy





Super Admin Role

Capabilities:

- Create, edit, and delete Admin accounts
- View and manage all customers across the system
- Access complete system analytics and reports
- Set admin-level privileges and permissions
- Configure customer priority levels
- Manage global WhatsApp integration settings
- View all queries and system-wide data
- Configure system-wide settings and preferences
- Access complete audit logs
- Manage database backups and maintenance

Data Access:

- **Scope:** Entire system
- **Customers:** All customers from all admins
- **Events:** All events across the platform
- **Statistics:** System-wide metrics

UI Components:

- Super Admin dashboard with global statistics
- Admin management interface
- System settings panel
- Global analytics and reports

- Customer priority management
- Query management system

Account Requirements:

- Email ID (mandatory)
 - Mobile number (mandatory)
 - Strong password policy enforced
 - Cannot be deleted if it's the last super admin
-

Admin Role

Capabilities:

- Manage assigned customers only
- Create and customize event cards in Design Studio
- Upload PDF templates and CSV guest lists
- Generate demo cards for approval
- Batch generate personalized cards
- Send cards via WhatsApp
- Download generated PDFs
- Track delivery status for their customers
- Accept new client orders
- Request customer assignments from Super Admin
- Access analytics for their customers
- Manage customer profiles and event details

Data Access:

- **Scope:** Only assigned customers
- **Customers:** Customers assigned by Super Admin
- **Events:** Events for assigned customers only
- **Statistics:** Metrics for their customers only

Limitations:

- Cannot see other admins' customers
- Cannot access system-wide settings
- Cannot create other admin accounts
- Cannot change their own role or privileges

UI Components:

- Admin dashboard (filtered to their customers)

- Design Studio with full functionality
- Customer Tab for their assigned customers
- Analytics limited to their data
- Settings for personal preferences

Account Requirements:

- Email ID (mandatory - cannot function without)
- Mobile number (mandatory)
- Must be approved by Super Admin
- Requires email verification
- Can reset password via email only

Special Features:

- Can help customers with password recovery (through email verification)
 - Can request approval for new customer assignments
 - Notified of new inquiries from potential customers
-

Customer Role

Capabilities:

- View their own profile and event details
- Submit inquiries online or offline
- Approve or reject demo cards
- View their guest lists
- Track card delivery status
- Provide feedback on designs
- Access their own RSVP data

Data Access:

- **Scope:** Own data only
- **Events:** Their events only
- **Guest Lists:** Their guests only
- **Statistics:** Their event metrics only

UI Components:

- Customer portal (simplified dashboard)
- Demo approval interface
- Guest list viewer
- Event status tracker

- Profile settings

Account Requirements:

- Mobile number (for WhatsApp communication)
- Email (optional but recommended)
- Can be managed by assigned admin

Interaction Model:

- Cannot access Design Studio directly
- Cannot upload files directly
- All actions facilitated through assigned Admin
- Receives WhatsApp notifications for key events

Authentication Requirements by Role

Role	Email	Mobile	Email Verification	Admin Approval
Super Admin	✓ Required	✓ Required	✓ Yes	N/A (first user)
Admin	✓ Required	✓ Required	✓ Yes	✓ Yes
Customer	⚠ Optional	✓ Required	⚠ If provided	✓ Yes

Access Control Matrix

Feature/Module	Super Admin	Admin	Customer
Dashboard - Global Statistics	✓ Full	✗ No	✗ No
Dashboard - Filtered Statistics	✓ Yes	✓ Yes	✓ Limited
User Management	✓ Full	✗ No	✗ No
Customer Management - All	✓ Full	✗ No	✗ No
Customer Management - Assigned	✓ Yes	✓ Yes	✗ No
Design Studio	✓ Yes	✓ Yes	✗ No
Customer Tab	✓ All	✓ Assigned	✗ No
PDF Generation	✓ Yes	✓ Yes	✗ No
WhatsApp Sending	✓ Yes	✓ Yes	✗ No
Demo Approval	⚠ View only	⚠ View only	✓ Yes

Feature/Module	Super Admin	Admin	Customer
Settings - Global	✓ Yes	✗ No	✗ No
Settings - Personal	✓ Yes	✓ Yes	✓ Limited
Analytics - Global	✓ Yes	✗ No	✗ No
Analytics - Filtered	✓ Yes	✓ Yes	✓ Limited
Query Management	✓ Yes	⚠ View assigned	✗ No
Audit Logs	✓ Full	⚠ Own actions	✗ No

Legend:

- ✓ Full Access
 - ⚠ Partial/Limited Access
 - ✗ No Access
-

Security & Data Isolation

Super Admin Isolation

- Access to all data through role verification
- Actions logged in audit trail
- Cannot be impersonated by other roles

Admin Isolation

- Customer Filter:** Queries automatically filtered by `assigned_admin_id`
- Event Filter:** Can only see events for their customers
- Guest List Filter:** Limited to guests of their customers' events
- Statistics:** Computed only for their customer base

Example SQL Pattern:

```
-- Admin sees only their customers
SELECT * FROM customers WHERE assigned_admin_id = ?
```

```
-- Admin sees only their customers' events
SELECT e.* FROM events e
JOIN customers c ON e.customer_id = c.id
WHERE c.assigned_admin_id = ?
```

```
-- Admin sees only their customers' guests
SELECT g.* FROM guests g
JOIN events e ON g.event_id = e.id
JOIN customers c ON e.customer_id = c.id
WHERE c.assigned_admin_id = ?
```

Customer Isolation

- Customers can only access their own data
 - Identified by `customer_id` in all queries
 - No cross-customer data visibility
 - Profile changes must be approved by admin
-

Role-Based UI Rendering

```
// Frontend role-based navigation
const navigation = {
  sudo_admin: [
    'Dashboard (Global)',
    'Admin Management',
    'All Customers',
    'Design Studio',
    'All Customer Tabs',
    'Analytics (Global)',
    'Settings (Global)',
    'Query Management'
  ],
  admin: [
    'Dashboard (Filtered)',
    'My Customers',
    'Design Studio',
    'Customer Tab (My Customers)',
    'Analytics (My Data)',
    'Settings (Personal)'
  ],
  customer: [
    'My Dashboard',
    'My Events',
    'Demo Approvals',
    'Guest Lists',
    'Profile Settings'
  ]
}
```

```
]
};
```

Project Structure

Enhanced Project Structure

```
Project_B/
|
└── backend/                               # FastAPI Backend
    ├── __init__.py                         # Package initialization
    └── main.py                            # Application entry point &
                                             # Router config
        ├── config.py                      # Configuration management (NEW)
        |
        └── auth/                           # Authentication Module
            ├── __init__.py
            ├── auth.py                     # JWT & password handling
            └── permissions.py             # Role-based access control
                                             # (NEW)
            └── password_reset.py          # Password recovery via email
                                             # (NEW)
|
└── database/                             # Database Layer
    ├── __init__.py
    ├── database.py
    ├── models.py
    └── migrations.py                      # SQLAlchemy models (optional)
                                              # Schema migration scripts (NEW)
|
└── routes/                               # API Routes
    ├── __init__.py
    ├── users.py                          # User management APIs (NEW)
    ├── customers.py                     # Customer CRUD APIs (ENHANCED)
    ├── events.py                        # Event management APIs (NEW)
    └── design_studio.py                # Design studio APIs (routes.py
                                             # refactored)
        ├── customer_tab.py            # Customer Tab APIs (NEW)
        └── whatsapp.py              # WhatsApp integration APIs
                                             # (NEW)
    └── analytics.py                   # Analytics & dashboard
```

(ENHANCED)

```
|   |   └── notifications.py          # Notification APIs (NEW)
|   |   └── settings.py             # Settings management (NEW)
|   |   └── queries.py              # Customer inquiry APIs (NEW)

|   └── services/
|       ├── __init__.py
|       ├── user_service.py          # User business logic
|       ├── customer_service.py     # Customer business logic
|       ├── event_service.py        # Event workflow management
|       ├── demo_approval_service.py # Demo approval workflow
|       ├── whatsapp_service.py     # WhatsApp sending logic
|       ├── pdf_service.py          # PDF generation orchestration
|       ├── notification_service.py # Notification dispatch
|       └── conflict_detection.py   # Zone conflict prevention

|   └── utils/
|       ├── __init__.py
|       ├── csv_parser.py           # CSV validation & parsing
|       ├── validators.py           # Input validation
|       ├── file_handler.py         # File upload/download
|       └── email_sender.py         # Email notification sender

└── tasks.py                         # Background tasks (if needed)

frontend/
|   ├── index.html                    # React Vite Frontend
|   ├── package.json                  # HTML entry point
|   ├── vite.config.js                # NPM dependencies
|   ├── tailwind.config.js            # Vite configuration
|   ├── postcss.config.js             # TailwindCSS config
|   └── PostCSS config

|   └── public/                       # Static assets
|       ├── logo.svg
|       └── favicon.ico

src/
|   ├── main.jsx                      # React entry point
|   ├── App.jsx                       # Main app with routing
|   └── index.css                     # TailwindCSS + custom styles
```

```
| contexts/
|   └── AuthContext.jsx          # Auth state (refactored)
|   └── UserContext.jsx         # User info & role (NEW)
|   └── NotificationContext.jsx # Global notifications (NEW)

| components/                      # Reusable Components (NEW)
|   └── common/
|     └── Button.jsx
|     └── Modal.jsx
|     └── Table.jsx
|     └── Select.jsx
|     └── FileUpload.jsx
|     └── StatusBadge.jsx

|   └── layout/
|     └── Header.jsx
|     └── Sidebar.jsx
|     └── Footer.jsx

|   └── charts/
|     └── LineChart.jsx
|     └── PieChart.jsx
|     └── BarChart.jsx

| pages/                           # Page Components
|   └── auth/
|     └── Login.jsx               # Login page
|     └── ForgotPassword.jsx    # Password reset (NEW)
|     └── Register.jsx          # Admin registration (NEW)

|   └── dashboard/
|     └── SuperAdminDashboard.jsx # Global stats (NEW)
|     └── AdminDashboard.jsx      # Filtered stats (NEW)
|     └── CustomerDashboard.jsx  # Customer view (NEW)

|   └── users/
|     └── UserManagement.jsx    # User CRUD (NEW)
|     └── AdminList.jsx          # Admin management (NEW)

|   └── customers/
|     └── CustomerList.jsx      # Customer list (ENHANCED)
```

```
|   |   |   └── CustomerForm.jsx          # Create/edit customer (NEW)
|   |   └── CustomerDetails.jsx        # Customer profile (NEW)
|
|   └── events/
|       ├── EventList.jsx            # Event listing (NEW)
|       └── EventDetails.jsx        # Event management (NEW)
|
|   └── design-studio/
|       |   ├── DesignStudio.jsx      # Main design interface
|(ENHANCED)
|       |   ├── ZoneEditor.jsx        # Zone drawing (NEW)
|       |   ├── CSVMapper.jsx        # CSV column mapping (NEW)
|       |   └── DemoGenerator.jsx     # Demo creation (NEW)
|
|   └── customer-tab/
|       ├── CustomerTab.jsx         # Guest management (NEW)
|       ├── GuestTable.jsx         # Guest list table (NEW)
|       ├── PDFViewer.jsx          # PDF preview (NEW)
|       └── BulkActions.jsx        # Select all/send all (NEW)
|
|   └── analytics/
|       ├── Analytics.jsx          # Charts page (ENHANCED)
|       └── Reports.jsx            # Report viewer
|
|   └── settings/
|       ├── GlobalSettings.jsx     # System settings (NEW)
|       ├── PersonalSettings.jsx   # User preferences (NEW)
|       └── WhatsAppConfig.jsx     # WhatsApp setup (NEW)
|
|   └── queries/
|       ├── QueryList.jsx          # Inquiry management (NEW)
|       └── QueryDetails.jsx       # Inquiry details (NEW)
|
|   └── hooks/
|       ├── useAuth.js             # Custom Hooks (NEW)
|       ├── useRole.js              # Auth hook
|       ├── useNotification.js     # Role check hook
|       └── useFetch.js             # Notification hook
|           # API fetch hook
|
|   └── utils/
|       └── api.js                 # Frontend Utilities (NEW)
|           # API client
```

```
|   └── constants.js                      # App constants
|   └── formatters.js                     # Data formatters
|   └── validators.js                    # Form validators
|
|── pdf_personalizer/                  # PDF Processing Engine
|   ├── __init__.py
|   └── core/
|       ├── __init__.py
|       ├── overlayer.py                 # Text overlay on PDF
|       ├── masking.py                  # Background masking
|       ├── generator.py                # Batch PDF generation
|       └── conflict_detector.py        # Zone conflict detection (NEW)
|
|── storage/                           # File Storage
|   ├── uploads/                        # Uploaded files
|   |   └── {customer_id}/
|   |       ├── {event_id}/
|   |       |   ├── template.pdf
|   |       |   └── guests.csv
|   |       └── ...
|   |
|   └── previews/                      # PDF preview images
|       └── {customer_id}/
|           ├── {event_id}/
|           |   ├── page_1.png
|           |   └── ...
|           └── ...
|
|   └── outputs/                        # Generated PDFs
|       └── {customer_id}/
|           ├── {event_id}/
|           |   ├── demo.pdf
|           |   ├── {guest_id}_card.pdf
|           |   └── ...
|           └── ...
|
|   └── temp/                            # Temporary files (NEW)
|
|── logs/                               # Application Logs (NEW)
|   ├── app.log                          # General application logs
|   └── error.log                        # Error logs
```

```

|   └─ whatsapp.log                                # WhatsApp API logs
|   └─ audit.log                                   # User action audit trail

|
└─ tests/                                         # Test Suite (NEW)
    ├─ backend/
    |   ├─ test_auth.py
    |   ├─ test_users.py
    |   ├─ test_customers.py
    |   └─ ...
    └─ frontend/
        └─ ...

|
└─ docs/                                          # Documentation (NEW)
    ├─ API.md                                       # API documentation
    ├─ DEPLOYMENT.md                               # Deployment guide
    ├─ USER_GUIDE.md                               # User manual
    └─ CONTRIBUTING.md                            # Contribution guidelines

|
└─ scripts/                                       # Utility Scripts (NEW)
    ├─ seed_database.py                           # Sample data seeder
    ├─ backup_database.sh                         # Database backup
    └─ cleanup_old_files.py                      # File cleanup cron

|
└─ .env.example                                  # Environment variables template
(NEW)
└─ .gitignore                                     # Git ignore rules
└─ invitewala.db                                 # SQLite database
└─ requirements.txt                               # Python dependencies
└─ requirements-dev.txt                          # Dev dependencies (NEW)
└─ README.md                                      # This documentation

```

Directory Purpose Summary

Directory	Purpose	Access Level
backend/routes/	API endpoints	All authenticated users
backend/services/	Business logic	Role-specific
frontend/src/pages/	UI pages	Role-based rendering
storage/uploads/	Original files	Admin who uploaded
storage/outputs/	Generated PDFs	Associated customer

Directory	Purpose	Access Level
logs/	System logs	Super Admin only

Core Features & Modules

Module Overview

Core Platform Modules

1. Authentication & Authorization
 - Multi-role JWT authentication
 - Email verification
 - Password reset workflow
 - Role-based access control
2. User Management (Super Admin Only)
 - Create/edit/delete Admin accounts
 - Set admin privileges
 - View all users
 - Manage customer assignments
3. Customer Management
 - Customer CRUD operations
 - Customer assignment to admins
 - Customer priority settings
 - Customer inquiry handling
 - Customer profile management
4. Event Management
 - Create events per customer
 - Store event details (date, type, venue)
 - Track event status
 - Event-specific guest lists
5. Design Studio
 - PDF template upload
 - Visual zone drawing (drag & drop)

- Zone configuration (text, font, color, alignment)
- CSV column mapping
- Zone conflict detection & prevention
- Preview generation
- Demo card creation

6. Demo Approval Workflow

- Generate single demo card
- Send demo to customer (WhatsApp/Email)
- Customer approval interface
- Feedback collection
- Revision tracking

7. Customer Tab (Guest Management)

- View all guests from CSV in interactive table
- Table columns: Name, WhatsApp No, PDF Link, Type, Status
- Guest list table with clickable PDF download links
- Individual guest selection via checkboxes
- Select all guests functionality
- PDF preview per guest (click to download)
- Download individual PDFs with proper filename
- Download selected PDFs as ZIP
- Download all PDFs (bulk download)
- Send via WhatsApp (individual/bulk)
- Resend button for failed/error guests
- Send custom reminder messages to selected guests
- Send thank you messages to selected guests
- Track send status per guest with error details
- Duplicate send prevention
- Advanced filtering dropdown (by status, invitation type, errors)
- Filter by: All, Pending, Sent, Delivered, Failed, Error
- Filter by invitation type from CSV data
- Error type categorization and display
- Generate comprehensive event reports (PDF/Excel/CSV)
- Export filtered guest lists
- Real-time statistics by filter

8. PDF Management

- Batch PDF generation
- PDF storage with naming convention
- PDF retrieval and display

- Auto-cleanup after event
- Trash/archive functionality

9. WhatsApp Integration

- WhatsApp Cloud API integration
- Send PDFs as attachments
- Custom message templates
- Delivery status tracking
- Error handling & retry logic
- Rate limiting

10. Dashboard & Analytics

- Total members/customers statistics
- Cards sent count
- Delivery success rate
- Active events tracking
- RSVP response charts
- Revenue/sales tracking (optional)

11. Notification System

- New customer assignment alerts
- Demo approval requests
- Delivery status updates
- Event milestone notifications
- System alerts

12. Settings & Configuration

- Global settings (Super Admin)
- WhatsApp API configuration
- Email SMTP settings
- User preferences
- Theme customization

Backend Components (Detailed)

1. main.py - Application Entry Point

Purpose: Initializes FastAPI application, configures middleware, includes routers, and serves static files.

Key Responsibilities:

- Create FastAPI app instance with metadata
- Configure CORS for cross-origin requests
- Mount static file directories for uploads and frontend
- Include all API routers with prefixes
- Setup exception handlers
- Configure logging
- SPA catch-all for React Router

Code Structure:

```
from fastapi import FastAPI, Request
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from backend.auth.auth import router as auth_router
from backend.routes.users import router as users_router
from backend.routes.customers import router as customers_router
from backend.routes.events import router as events_router
from backend.routes.design_studio import router as design_router
from backend.routes.customer_tab import router as customer_tab_router
from backend.routes.whatsapp import router as whatsapp_router
from backend.routes.analytics import router as analytics_router
from backend.routes.notifications import router as notifications_router
from backend.routes.settings import router as settings_router
from backend.routes.queries import router as queries_router
from backend.database.database import init_db
import logging

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('logs/app.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)
```

```
# Create FastAPI app
app = FastAPI(
    title="Invitewala Platform API",
    description="Multi-user event card management and distribution system",
    version="2.0.0",
    docs_url="/api/docs",
    redoc_url="/api/redoc"
)

# Configure CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Configure appropriately for production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Initialize database
@app.on_event("startup")
async def startup_event():
    logger.info("Starting Invitewala Platform...")
    init_db()
    logger.info("Database initialized successfully")

# Global exception handler
@app.exception_handler(Exception)
async def global_exception_handler(request: Request, exc: Exception):
    logger.error(f"Global exception: {exc}", exc_info=True)
    return JSONResponse(
        status_code=500,
        content={"detail": "Internal server error"}
    )

# Include routers with prefixes
app.include_router(auth_router, prefix="/api/auth", tags=["Authentication"])
app.include_router(users_router, prefix="/api/users", tags=["Users"])
app.include_router(customers_router, prefix="/api/customers", tags=["Customers"])
app.include_router(events_router, prefix="/api/events", tags=["Events"])
app.include_router(design_router, prefix="/api/design", tags=["Design Structures"])
app.include_router(customer_tab_router, prefix="/api/customer-tab", tags=["Customer Tab"])
app.include_router(whatsapp_router, prefix="/api/whatsapp", tags=["WhatsApp"])
app.include_router(analytics_router, prefix="/api/analytics", tags=["Analytics"])
```

```

app.include_router(notifications_router, prefix="/api/notifications", tags=["Notifications"])
app.include_router(settings_router, prefix="/api/settings", tags=["Settings"])
app.include_router(queries_router, prefix="/api/queries", tags=["Queries"])

# Mount static files
app.mount("/storage", StaticFiles(directory="storage"), name="storage")

# Serve React frontend (SPA)
@app.get("/{full_path:path}")
async def serve_spa(full_path: str):
    """Catch-all route for React Router"""
    return StaticFiles(directory="frontend/dist", html=True)

# Health check
@app.get("/api/health")
async def health_check():
    return {"status": "healthy", "version": "2.0.0"}

```

Static File Serving:

Path	Directory	Purpose
/storage	storage/	Uploaded templates, generated PDFs, previews
{any}	frontend/dist	Built React SPA (fallback for client-side routing)

Environment Configuration:

```

# config.py
import os
from pydantic import BaseSettings

class Settings(BaseSettings):
    DATABASE_URL: str = "sqlite:///./invitewala.db"
    SECRET_KEY: str = os.getenv("SECRET_KEY", "your-secret-key-change-in-ACCESS_TOKEN_EXPIRE_HOURS: int = 24

    # WhatsApp Configuration
    WHATSAPP_API_URL: str = os.getenv("WHATSAPP_API_URL")
    WHATSAPP_API_TOKEN: str = os.getenv("WHATSAPP_API_TOKEN")
    WHATSAPP_PHONE_NUMBER_ID: str = os.getenv("WHATSAPP_PHONE_NUMBER_ID")

    # Email Configuration
    SMTP_HOST: str = os.getenv("SMTP_HOST", "smtp.gmail.com")

```

```

SMTP_PORT: int = int(os.getenv("SMTP_PORT", "587"))
SMTP_USERNAME: str = os.getenv("SMTP_USERNAME", "")
SMTP_PASSWORD: str = os.getenv("SMTP_PASSWORD", "")
SMTP_FROM_EMAIL: str = os.getenv("SMTP_FROM_EMAIL", "noreply@invitewa.com")

# File Storage
UPLOAD_DIR: str = "storage/uploads"
OUTPUT_DIR: str = "storage/outputs"
PREVIEW_DIR: str = "storage/previews"
MAX_UPLOAD_SIZE: int = 10 * 1024 * 1024 # 10MB

# PDF Generation
PDF_DPI: int = 150
PDF_QUALITY: int = 90

class Config:
    env_file = ".env"

settings = Settings()

```

2. auth/auth.py - Authentication System

Purpose: Comprehensive authentication system with JWT tokens, password hashing, email verification, and role-based access.

Key Functions:

Password Management

```

import bcrypt
import jwt
from datetime import datetime, timedelta
from fastapi import HTTPException, Depends, Header
from backend.config import settings

def hash_password(password: str) -> str:
    """Hash a password using bcrypt with salt"""
    salt = bcrypt.gensalt()
    return bcrypt.hashpw(password.encode('utf-8'), salt).decode('utf-8')

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify a password against its hash"""

```

```
        return bcrypt.checkpw(  
            plain_password.encode('utf-8'),  
            hashed_password.encode('utf-8'))  
)
```

JWT Token Management

```
def create_access_token(data: dict, expires_delta: timedelta = None) -> str:  
    """  
    Create a JWT access token with user data  
  
    Args:  
        data: User information to encode  
        expires_delta: Token expiration time  
  
    Returns:  
        JWT token string  
    """  
    to_encode = data.copy()  
    expire = datetime.utcnow() + (expires_delta or timedelta(hours=settings.JWT_EXPIRE_HOURS))  
    to_encode.update({"exp": expire, "iat": datetime.utcnow()})  
    encoded_jwt = jwt.encode(to_encode, settings.SECRET_KEY, algorithm="HS256")  
    return encoded_jwt  
  
def verify_token(token: str) -> dict:  
    """  
    Verify and decode a JWT token  
  
    Args:  
        token: JWT token string  
  
    Returns:  
        Decoded token payload  
  
    Raises:  
        HTTPException: If token is invalid or expired  
    """  
    try:  
        payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])  
        return payload  
    except jwt.ExpiredSignatureError:  
        raise HTTPException(status_code=401, detail="Token has expired")
```

```
except jwt.InvalidTokenError:  
    raise HTTPException(status_code=401, detail="Invalid token")
```

User Authentication Dependency

```
async def get_current_user(authorization: str = Header(None)) -> dict:  
    """  
    FastAPI dependency to extract current user from Authorization header  
  
    Args:  
        authorization: Authorization header value (Bearer <token>)  
  
    Returns:  
        User data from token  
  
    Raises:  
        HTTPException: If authorization fails  
    """  
  
    if not authorization or not authorization.startswith("Bearer "):  
        raise HTTPException(status_code=401, detail="Missing or invalid auth token")  
  
    token = authorization.split(" ")[1]  
    payload = verify_token(token)  
  
    # Fetch user from database  
    with get_cursor() as cursor:  
        cursor.execute("SELECT id, email, name, role FROM users WHERE id = %s", (token,))  
        user = cursor.fetchone()  
        if not user:  
            raise HTTPException(status_code=401, detail="User not found")  
  
    return dict(user)
```

Role-Based Access Control

```
def require_role(*allowed_roles: str):  
    """  
    Dependency factory for role-based access control  
  
    Usage:  
        @app.get("/admin-only")  
        async def admin_endpoint(user: dict = Depends(require_role("sudo"))):  
            ...  
    """
```

...

Args:

 allowed_roles: Tuple of allowed role names

Returns:

 FastAPI dependency function

"""

```
async def role_checker(user: dict = Depends(get_current_user)) -> dict:
    if user["role"] not in allowed_roles:
        raise HTTPException(
            status_code=403,
            detail=f"Access forbidden. Required role: {', '.join(allowed_roles)}")
    return user
return role_checker
```

API Endpoints:

POST /api/auth/login

Login with email and password.

Request:

```
{
    "email": "admin@invitewala.com",
    "password": "admin123"
}
```

Response (200):

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIs...",
    "token_type": "bearer",
    "user": {
        "id": 1,
        "email": "admin@invitewala.com",
        "name": "John Doe",
        "role": "admin",
        "mobile": "+919998118489"
    }
}
```

Implementation:

```
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel

router = APIRouter()

class LoginRequest(BaseModel):
    email: str
    password: str

@router.post("/login")
async def login(request: LoginRequest):
    """Authenticate user and return JWT token"""
    with get_cursor() as cursor:
        cursor.execute(
            "SELECT * FROM users WHERE email = ?",
            (request.email,)
        )
        user = cursor.fetchone()

        if not user or not verify_password(request.password, user["password"]):
            raise HTTPException(status_code=401, detail="Invalid email or password")

        if not user["is_verified"]:
            raise HTTPException(status_code=403, detail="Email not verified")

    token_data = {
        "user_id": user["id"],
        "email": user["email"],
        "role": user["role"]
    }
    access_token = create_access_token(token_data)

    return {
        "access_token": access_token,
        "token_type": "bearer",
        "user": {
            "id": user["id"],
            "email": user["email"],
            "name": user["name"],
            "role": user["role"],
            "mobile": user["mobile"]
        }
    }
```

```
    }  
}
```

POST /api/auth/register

Register new admin account (requires Super Admin approval).

Request:

```
{  
  "email": "newadmin@example.com",  
  "password": "SecurePass123!",  
  "name": "Jane Smith",  
  "mobile": "+919876543210",  
  "role": "admin"  
}
```

Response (201):

```
{  
  "message": "Registration successful. Awaiting Super Admin approval.",  
  "user_id": 5,  
  "verification_email_sent": true  
}
```

GET /api/auth/me

Get current authenticated user information.

Response (200):

```
{  
  "id": 2,  
  "email": "admin@invitewala.com",  
  "name": "John Doe",  
  "role": "admin",  
  "mobile": "+919998118489",  
  "created_at": "2024-01-15T10:30:00Z",  
  "is_verified": true,  
  "assigned_customers_count": 12  
}
```

POST /api/auth/forgot-password

Request password reset via email.

Request:

```
{  
  "email": "admin@invitewala.com"  
}
```

Response (200):

```
{  
  "message": "Password reset email sent"  
}
```

POST /api/auth/reset-password

Reset password using token from email.

Request:

```
{  
  "token": "reset-token-from-email",  
  "new_password": "NewSecurePass123!"  
}
```

Response (200):

```
{  
  "message": "Password reset successful"  
}
```

POST /api/auth/verify-email

Verify email address using token.

Request:

```
{  
  "token": "verification-token-from-email"  
}
```

Response (200):

```
{  
    "message": "Email verified successfully"  
}
```

3. routes/users.py - User Management (NEW)

Purpose: CRUD operations for user accounts, accessible only to Super Admin.

Key Endpoints:

GET /api/users

List all users with filtering.

Query Parameters:

- `role` : Filter by role (sudo_admin, admin, customer)
- `search` : Search by name or email
- `limit` : Number of results (default: 50)
- `offset` : Pagination offset

Response (200):

```
{  
    "users": [  
        {  
            "id": 2,  
            "email": "admin1@example.com",  
            "name": "John Doe",  
            "role": "admin",  
            "mobile": "+919998118489",  
            "is_verified": true,  
            "created_at": "2024-01-15T10:30:00Z",  
            "assigned_customers_count": 12  
        },  
        {  
            "id": 3,  
            "email": "admin2@example.com",  
            "name": "Jane Smith",  
            "role": "customer",  
            "mobile": null,  
            "is_verified": false,  
            "created_at": "2024-01-15T10:30:00Z",  
            "assigned_customers_count": 0  
        }  
    ]  
}
```

```
        "role": "admin",
        "mobile": "+919876543210",
        "is_verified": false,
        "created_at": "2024-02-01T14:20:00Z",
        "assigned_customers_count": 0
    }
],
"total": 2,
"limit": 50,
"offset": 0
}
```

POST /api/users

Create a new admin account (Super Admin only).

Request:

```
{
    "email": "newadmin@example.com",
    "password": "SecurePass123!",
    "name": "New Admin",
    "mobile": "+919123456789",
    "role": "admin"
}
```

Response (201):

```
{
    "message": "Admin created successfully",
    "user": {
        "id": 5,
        "email": "newadmin@example.com",
        "name": "New Admin",
        "role": "admin",
        "mobile": "+919123456789"
    }
}
```

PUT /api/users/{user_id}

Update user details.

Request:

```
{  
    "name": "Updated Name",  
    "mobile": "+919999999999",  
    "is_verified": true  
}
```

Response (200):

```
{  
    "message": "User updated successfully",  
    "user": { ... }  
}
```

DELETE /api/users/{user_id}

Delete a user account.

Response (200):

```
{  
    "message": "User deleted successfully"  
}
```

POST /api/users/{user_id}/approve

Approve a pending admin registration.

Response (200):

```
{  
    "message": "Admin approved successfully",  
    "verification_email_sent": true  
}
```

4. routes/customers.py - Customer Management (ENHANCED)

Purpose: CRUD operations for customer records with admin assignment tracking.

Key Endpoints:

GET /api/customers

List customers (filtered by role).

Query Parameters:

- `search` : Search by name, email, or phone
- `assigned_admin_id` : Filter by assigned admin (Super Admin only)
- `limit` : Number of results
- `offset` : Pagination offset

Response (200):

```
{  
    "customers": [  
        {  
            "id": 1,  
            "first_name": "Raj",  
            "last_name": "Patel",  
            "email": "raj@example.com",  
            "phone": "+919998118489",  
            "address": "123 Main St, Surat, Gujarat",  
            "notes": "Wedding in December",  
            "assigned_admin_id": 2,  
            "assigned_admin_name": "John Doe",  
            "priority": "high",  
            "created_at": "2024-01-10T08:00:00Z",  
            "event_count": 2  
        }  
    ],  
    "total": 1  
}
```

Role-Based Filtering:

- **Super Admin**: Sees all customers
- **Admin**: Sees only customers where `assigned_admin_id = current_user.id`
- **Customer**: Only their own profile

POST /api/customers

Create a new customer.

Request:

```
{  
    "first_name": "Priya",  
    "last_name": "Shah",  
    "email": "priya@example.com",  
    "phone": "+919876543210",  
    "address": "456 Park Ave, Surat",  
    "notes": "Anniversary event",  
    "assigned_admin_id": 2,  
    "priority": "medium"  
}
```

Response (201):

```
{  
    "message": "Customer created successfully",  
    "customer": {  
        "id": 10,  
        "first_name": "Priya",  
        "last_name": "Shah",  
        ...  
    }  
}
```

PUT /api/customers/{customer_id}

Update customer details.

Authorization:

- Super Admin can update any customer
- Admin can only update their assigned customers

Request:

```
{  
    "first_name": "Priya Updated",  
    "phone": "+919999999999",  
    "notes": "Updated notes"  
}
```

DELETE /api/customers/{customer_id}

Soft delete a customer (moves to archive).

POST /api/customers/{customer_id}/assign

Assign customer to an admin (Super Admin only).

Request:

```
{  
    "admin_id": 3  
}
```

Response (200):

```
{  
    "message": "Customer assigned successfully",  
    "customer_id": 10,  
    "admin_id": 3,  
    "admin_name": "Jane Smith"  
}
```

GET /api/customers/{customer_id}/events

Get all events for a specific customer.

Response (200):

```
{  
    "customer_id": 10,  
    "events": [  
        {  
            "id": 1,  
            "event_name": "Wedding Ceremony",  
            "event_type": "wedding",  
            "event_date": "2024-12-15",  
            "status": "active",  
            "guest_count": 250,  
            "cards_sent": 200,  
            "cards_delivered": 180  
        }  
    ]  
}
```

5. routes/events.py - Event Management (NEW)

Purpose: Manage events associated with customers.

Database Schema:

```
CREATE TABLE events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL REFERENCES customers(id),
    event_name TEXT NOT NULL,
    event_type TEXT CHECK(event_type IN ('wedding', 'birthday', 'anniversary')),
    event_date DATE,
    venue TEXT,
    notes TEXT,
    status TEXT CHECK(status IN ('pending', 'active', 'completed', 'cancelled')),
    template_pdf_path TEXT,
    guest_csv_path TEXT,
    zones_config TEXT, -- JSON string
    demo_approved BOOLEAN DEFAULT 0,
    demo_pdf_path TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP
) ;
```

Key Endpoints:

POST /api/events

Create a new event for a customer.

Request:

```
{
    "customer_id": 10,
    "event_name": "Raj & Priya Wedding",
    "event_type": "wedding",
    "event_date": "2024-12-15",
    "venue": "Grand Palace, Surat",
    "notes": "Traditional ceremony"
}
```

Response (201):

```
{  
    "message": "Event created successfully",  
    "event": {  
        "id": 1,  
        "customer_id": 10,  
        "event_name": "Raj & Priya Wedding",  
        "status": "pending",  
        ...  
    }  
}
```

GET /api/events/{event_id}

Get detailed event information.

Response (200):

```
{  
    "id": 1,  
    "customer_id": 10,  
    "customer_name": "Priya Shah",  
    "event_name": "Raj & Priya Wedding",  
    "event_type": "wedding",  
    "event_date": "2024-12-15",  
    "venue": "Grand Palace, Surat",  
    "status": "active",  
    "template_pdf_path": "/storage/uploads/10/1/template.pdf",  
    "guest_csv_path": "/storage/uploads/10/1/guests.csv",  
    "zones_config": {...},  
    "demo_approved": true,  
    "guest_count": 250,  
    "cards_generated": 250,  
    "cards_sent": 200,  
    "cards_delivered": 180,  
    "delivery_rate": 90.0,  
    "created_at": "2024-11-01T10:00:00Z"  
}
```

PUT /api/events/{event_id}

Update event details.

DELETE /api/events/{event_id}

Delete an event and all associated data.

Warning: This will delete:

- All generated PDFs
- Guest list data
- Delivery logs
- Demo approvals

Response (200):

```
{  
    "message": "Event deleted successfully",  
    "files_deleted": 250,  
    "guests_deleted": 250  
}
```

POST /api/events/{event_id}/complete

Mark an event as completed.

Effect:

- Status changes to "completed"
- Triggers auto-cleanup process
- Archives data
- Deletes generated PDFs (optional)

Request:

```
{  
    "delete_pdfs": true,  
    "archive_data": true  
}
```

Response (200):

```
{  
    "message": "Event completed successfully",  
    "pdfs_deleted": 250,  
    "data_archived": true  
}
```

6. routes/design_studio.py - Design Studio APIs (REFACTORED)

Purpose: Handles PDF template upload, zone definition, CSV mapping, and demo generation.

Key Endpoints:

POST /api/design/upload-template

Upload PDF template for an event.

Request (multipart/form-data):

```
event_id: 1  
file: template.pdf (binary)
```

Response (201):

```
{  
    "message": "Template uploaded successfully",  
    "event_id": 1,  
    "template_path": "/storage/uploads/10/1/template.pdf",  
    "page_count": 2,  
    "previews": [  
        "/storage/previews/10/1/page_1.png",  
        "/storage/previews/10/1/page_2.png"  
    ]  
}
```

POST /api/design/upload-csv

Upload CSV guest list for an event.

Request (multipart/form-data):

```
event_id: 1  
file: guests.csv (binary)
```

CSV Format:

```
name,phone,invitation_type,table_number,message  
Raj Patel,+919998118489,VIP,A1,Looking forward to celebrating with you!  
Priya Shah,+919876543210,Standard,B5,
```

Response (201):

```
{
  "message": "CSV uploaded successfully",
  "event_id": 1,
  "csv_path": "/storage/uploads/10/1/guests.csv",
  "guest_count": 250,
  "columns": ["name", "phone", "invitation_type", "table_number", "mess
  "sample_rows": [
    {
      "name": "Raj Patel",
      "phone": "+919998118489",
      "invitation_type": "VIP",
      "table_number": "A1",
      "message": "Looking forward to celebrating with you!"
    }
  ]
}
```

POST /api/design/save-zones

Save zone definitions for an event.

Request:

```
{
  "event_id": 1,
  "zones": [
    {
      "id": "zone1",
      "page": 0,
      "x": 100,
      "y": 200,
      "width": 300,
      "height": 50,
      "csv_column": "name",
      "font_family": "Arial",
      "font_size": 24,
      "font_color": "#000000",
      "alignment": "center",
      "mask_mode": "auto_sample",
      "mask_color": null
    },
    {
      "id": "zone2",
      "page": 1,
      "x": 200,
      "y": 300,
      "width": 200,
      "height": 50,
      "csv_column": "phone",
      "font_family": "Times New Roman",
      "font_size": 18,
      "font_color": "#FF0000",
      "alignment": "left",
      "mask_mode": "auto_sample",
      "mask_color": null
    }
  ]
}
```

```

        "page": 0,
        "x": 100,
        "y": 300,
        "width": 300,
        "height": 40,
        "csv_column": "invitation_type",
        "font_family": "Times New Roman",
        "font_size": 18,
        "font_color": "#333333",
        "alignment": "left",
        "mask_mode": "overlay",
        "mask_color": null
    }
]
}

```

Response (200):

```
{
    "message": "Zones saved successfully",
    "event_id": 1,
    "zone_count": 2,
    "conflicts_detected": false
}
```

Zone Conflict Detection:

The system checks for overlapping zones and returns warnings:

```
{
    "message": "Zones saved with warnings",
    "zone_count": 2,
    "conflicts_detected": true,
    "conflicts": [
        {
            "zone1_id": "zone1",
            "zone2_id": "zone2",
            "overlap_area": 150,
            "recommendation": "Adjust zone2 position to avoid overlap"
        }
    ]
}
```

POST /api/design/generate-demo

Generate a single demo PDF for customer approval.

Request:

```
{  
    "event_id": 1,  
    "sample_row_index": 0  
}
```

Process:

1. Load template PDF
2. Load first row of CSV (or specified row)
3. Apply all zones with text from CSV
4. Save demo PDF
5. Return demo PDF path

Response (200):

```
{  
    "message": "Demo generated successfully",  
    "event_id": 1,  
    "demo_pdf_path": "/storage/outputs/10/1/demo.pdf",  
    "preview_image": "/storage/previews/10/1/demo_preview.png",  
    "sample_data_used": {  
        "name": "Raj Patel",  
        "phone": "+919998118489",  
        "invitation_type": "VIP"  
    }  
}
```

POST /api/design/generate-batch

Generate personalized PDFs for all guests after demo approval.

Request:

```
{  
    "event_id": 1  
}
```

Process:

1. Verify demo is approved
2. Load template and zones
3. Load entire CSV
4. For each guest row:
 - o Create personalized PDF
 - o Save with naming convention: `{guest_id}_card.pdf`
 - o Store guest record in database
5. Return batch generation summary

Response (200):

```
{  
    "message": "Batch generation completed",  
    "event_id": 1,  
    "total_guests": 250,  
    "pdfs_generated": 250,  
    "generation_time_seconds": 45.3,  
    "output_directory": "/storage/outputs/10/1/",  
    "failed_guests": []  
}
```

Background Processing (for large batches):

```
{  
    "message": "Batch generation started",  
    "task_id": "batch_123456",  
    "status": "processing",  
    "estimated_time_seconds": 60  
}
```

Check status:

```
GET /api/design/batch-status/{task_id}
```

7. routes/customer_tab.py - Customer Tab APIs (NEW)

Purpose: Comprehensive guest list management with PDF downloads, WhatsApp sending, custom messaging (reminders/thank you), status tracking, filtering, and report generation.

Database Schema:

```

CREATE TABLE guests (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_id INTEGER NOT NULL REFERENCES events(id),
    name TEXT NOT NULL,
    phone TEXT,
    whatsapp_number TEXT, -- Dedicated WhatsApp number (may differ from
    invitation_type TEXT,
    csv_row_index INTEGER,
    csv_data TEXT, -- JSON string of full CSV row
    pdf_path TEXT,
    pdf_name TEXT, -- Human-readable PDF filename (e.g., "Raj_Patel_VIP_
    sent_status TEXT CHECK(sent_status IN ('pending', 'sent', 'delivered'
    sent_at TIMESTAMP,
    delivered_at TIMESTAMP,
    whatsapp_message_id TEXT,
    custom_message TEXT,
    error_code TEXT, -- Error code from WhatsApp API
    error_message TEXT, -- Detailed error description
    error_type TEXT, -- Categorized error type (invalid_number, blocked,
    reminder_sent BOOLEAN DEFAULT 0,
    reminder_sent_at TIMESTAMP,
    thankyou_sent BOOLEAN DEFAULT 0,
    thankyou_sent_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ;

```

Key Endpoints:

GET /api/customer-tab/events/{event_id}/guests

Get all guests for an event with their PDF and send status, with advanced filtering capabilities.

Query Parameters:

- `filter_status` : Filter by send status (all, pending, sent, delivered, failed, error)
- `filter_invitation_type` : Filter by invitation type from CSV
- `search` : Search by guest name or phone number
- `sort_by` : Sort by field (name, sent_at, status)
- `sort_order` : asc or desc
- `limit` : Number of results per page
- `offset` : Pagination offset

Response (200):

```
{  
    "event_id": 1,  
    "total_guests": 250,  
    "filtered_count": 20,  
    "filters_applied": {  
        "status": "error",  
        "invitation_type": null  
    },  
    "statistics": {  
        "total": 250,  
        "pending": 50,  
        "sent": 150,  
        "delivered": 180,  
        "failed": 15,  
        "error": 5,  
        "by_invitation_type": {  
            "VIP": {"total": 50, "sent": 45, "error": 2},  
            "Standard": {"total": 150, "sent": 140, "error": 3},  
            "Family": {"total": 50, "sent": 48, "error": 0}  
        }  
    },  
    "guests": [  
        {  
            "id": 1,  
            "name": "Raj Patel",  
            "phone": "+919998118489",  
            "whatsapp_number": "+919998118489",  
            "invitation_type": "VIP",  
            "pdf_path": "/storage/outputs/10/1/1_card.pdf",  
            "pdf_name": "Raj_Patel_VIP_Invitation.pdf",  
            "pdf_download_link": "/api/customer-tab/guests/1/download-pdf",  
            "sent_status": "delivered",  
            "sent_at": "2024-12-10T14:30:00Z",  
            "delivered_at": "2024-12-10T14:31:15Z",  
            "error_code": null,  
            "error_message": null,  
            "error_type": null,  
            "reminder_sent": false,  
            "thankyou_sent": false,  
            "csv_data": {  
                "name": "Raj Patel",  
                "phone": "+919998118489",  
                "invitation_type": "VIP",  
                "table_number": "A1",  
                "seating_order": 1  
            }  
        }  
    ]  
}
```

```

        "message": "Looking forward!",
    }
},
{
    "id": 3,
    "name": "Amit Kumar",
    "phone": "+911234567890",
    "whatsapp_number": "+911234567890",
    "invitation_type": "Standard",
    "pdf_path": "/storage/outputs/10/1/3_card.pdf",
    "pdf_name": "Amit_Kumar_Standard_Invitation.pdf",
    "pdf_download_link": "/api/customer-tab/guests/3/download-pdf",
    "sent_status": "error",
    "sent_at": "2024-12-10T14:32:00Z",
    "delivered_at": null,
    "error_code": "131026",
    "error_message": "Message undeliverable - Invalid phone number",
    "error_type": "invalid_number",
    "reminder_sent": false,
    "thankyou_sent": false,
    "csv_data": {
        "name": "Amit Kumar",
        "phone": "+911234567890",
        "invitation_type": "Standard",
        "table_number": "C10"
    }
}
]
}

```

Customer Tab Table Structure:

The guest table displays the following columns with interactive features:

Column	Display	Features
Checkbox	Select individual guest	Bulk selection for actions
Guest Name	Guest full name	Searchable, sortable
WhatsApp Number	Guest WhatsApp number	Click to view/edit, formatted display
PDF File	Generated PDF filename (clickable link)	Click to instantly download PDF

Column	Display	Features
Invitation Type	Type from CSV data	Filterable, color-coded badges
Action Status	Status with icon and error details	Color-coded: Green (sent/delivered), Red (error/failed), Yellow (pending)
Download Button	Individual download button	Downloads specific guest's PDF
Resend Button	Custom resend button	Only visible for failed/error status

Action Status Column Details:

- **Pending**: Yellow icon, "Pending" text
- **Sent**: Blue icon, "Sent" text, timestamp
- **Delivered**: Green checkmark icon, "Delivered ✓" text, timestamp
- **Failed**: Red X icon, "Failed X" text, error type in tooltip
- **Error**: Red warning icon, "Error" text, detailed error message on hover/click

Error Type Categories:

- `invalid_number` : Invalid or unregistered WhatsApp number
- `blocked` : User has blocked the business number
- `rate_limit` : Rate limit exceeded, retry later
- `network_error` : Network or connectivity issue
- `media_upload_failed` : PDF upload to WhatsApp failed
- `unknown` : Unknown error, see error message for details

GET /api/customer-tab/events/{event_id}/filter-options

Get available filter options for the event.

Response (200):

```
{
  "event_id": 1,
  "filter_options": {
    "status": [
      {"value": "all", "label": "All Guests", "count": 250},
      {"value": "pending", "label": "Pending", "count": 50},
      {"value": "sent", "label": "Sent", "count": 150},
      {"value": "delivered", "label": "Delivered", "count": 180},
      {"value": "failed", "label": "Failed", "count": 15}
    ]
  }
}
```

```

        {"value": "error", "label": "Error", "count": 5}
    ],
    "invitation_type": [
        {"value": "all", "label": "All Types", "count": 250},
        {"value": "VIP", "label": "VIP", "count": 50},
        {"value": "Standard", "label": "Standard", "count": 150},
        {"value": "Family", "label": "Family", "count": 50}
    ],
    "error_type": [
        {"value": "all", "label": "All Errors", "count": 5},
        {"value": "invalid_number", "label": "Invalid Number", "count": 1},
        {"value": "blocked", "label": "Blocked", "count": 1},
        {"value": "rate_limit", "label": "Rate Limited", "count": 1}
    ]
}
}
}

```

GET /api/customer-tab/guests/{guest_id}/download-pdf

Download a single guest's PDF (instant download).

Response: PDF file download with proper content-disposition header

- Filename: `{guest_name}_{invitation_type}_Invitation.pdf`
- Example: `Raj_Patel_VIP_Invitation.pdf`

POST /api/customer-tab/guests/download-selected

Download multiple guests' PDFs as a ZIP file.

Request:

```
{
  "guest_ids": [1, 2, 3, 4, 5]
}
```

Response: ZIP file download containing selected PDFs

- Filename: `Event_{event_name}_Invitations_{date}.zip`
- Example: `Event_Wedding_Invitations_20241210.zip`

POST /api/customer-tab/events/{event_id}/download-all

Download all guests' PDFs for an event as a ZIP file.

Query Parameters:

- `filter_status` : Optional filter (e.g., download only "pending" guests' PDFs)
- `filter_invitation_type` : Optional filter by invitation type

Response: ZIP file download

POST /api/customer-tab/guests/{guest_id}/send

Send invitation card to a single guest via WhatsApp with optional custom message.

Request:

```
{  
    "custom_message": "We're excited to invite you to our wedding!"  
}
```

Response (200):

```
{  
    "message": "Invitation sent successfully",  
    "guest_id": 1,  
    "guest_name": "Raj Patel",  
    "phone": "+919998118489",  
    "whatsapp_message_id": "wamid.HBgNOTE5OTk4MTE4NDg5FQIAERgS...",  
    "sent_at": "2024-12-10T14:30:00Z",  
    "status": "sent"  
}
```

POST /api/customer-tab/guests/{guest_id}/resend

Resend invitation to a guest (for failed/error status guests).

Request:

```
{  
    "custom_message": "Resending your invitation with corrected details",  
    "update_phone": "+919998118490" // Optional: update phone if it was  
}
```

Response (200):

```
{  
    "message": "Invitation resent successfully",  
    "guest_id": 3,  
    "previous_status": "error",  
    "new_status": "sent",  
    "attempts": 2  
}
```

POST /api/customer-tab/guests/send-selected

Send invitations to multiple selected guests.

Request:

```
{  
    "guest_ids": [1, 2, 3],  
    "custom_message": "Join us in celebrating!"  
}
```

Response (200):

```
{  
    "message": "Batch send initiated",  
    "total_guests": 3,  
    "successful": 2,  
    "failed": 1,  
    "results": [  
        {  
            "guest_id": 1,  
            "guest_name": "Raj Patel",  
            "status": "sent",  
            "phone": "+919998118489"  
        },  
        {  
            "guest_id": 2,  
            "guest_name": "Priya Shah",  
            "status": "sent",  
            "phone": "+919876543210"  
        },  
        {  
            "guest_id": 3,  
            "guest_name": "Amit Kumar",  
            "status": "error",  
            "phone": null  
        }  
    ]  
}
```

```

        "status": "failed",
        "phone": "+911234567890",
        "error": "Invalid phone number"
    }
]
}

```

POST /api/customer-tab/events/{event_id}/send-all

Send invitations to all guests (with pending status) or filtered guests.

Request:

```
{
    "custom_message": "You're invited!",
    "send_only_pending": true,
    "filter_invitation_type": "VIP" // Optional: send only to VIP guests
}
```

Response (200):

```
{
    "message": "Batch send initiated",
    "event_id": 1,
    "total_guests": 250,
    "already_sent": 200,
    "sending_now": 50,
    "task_id": "send_batch_123456",
    "status": "processing"
}
```

Duplicate Prevention Logic:

- System checks `sent_status` before sending
- If guest already has status "sent" or "delivered", skip them
- Only guests with "pending", "failed", or "error" status are eligible for resend

POST /api/customer-tab/guests/send-reminder

Send reminder messages to selected guests (custom message button).

Purpose: Send follow-up reminder messages to guests who have already received invitations.

Request:

```
{  
    "guest_ids": [1, 2, 3, 5, 8],  
    "reminder_message": "This is a gentle reminder about our wedding on I  
    "message_type": "reminder"  
}
```

Response (200):

```
{  
    "message": "Reminder sent to selected guests",  
    "total_guests": 5,  
    "successful": 5,  
    "failed": 0,  
    "reminder_type": "reminder",  
    "results": [  
        {  
            "guest_id": 1,  
            "guest_name": "Raj Patel",  
            "status": "sent",  
            "sent_at": "2024-12-11T10:00:00Z"  
        }  
    ]  
}
```

Database Updates:

- Sets `reminder_sent = true`
- Records `reminder_sent_at` timestamp
- Creates log entry in `whatsapp_logs` table

POST /api/customer-tab/guests/send-thankyou

Send thank you messages to selected guests (custom message button).

Purpose: Send appreciation messages to guests after the event.

Request:

```
{  
    "guest_ids": [1, 2, 3, 4, 5, 6, 7],  
    "thankyou_message": "Thank you so much for being part of our special
```

```
        "message_type": "thankyou"
    }
```

Response (200):

```
{
    "message": "Thank you messages sent to selected guests",
    "total_guests": 7,
    "successful": 7,
    "failed": 0,
    "message_type": "thankyou",
    "results": [
        {
            "guest_id": 1,
            "guest_name": "Raj Patel",
            "status": "sent",
            "sent_at": "2024-12-16T18:00:00Z"
        }
    ]
}
```

Database Updates:

- Sets `thankyou_sent = true`
- Records `thankyou_sent_at` timestamp
- Creates log entry in `whatsapp_logs` table

POST /api/customer-tab/events/{event_id}/send-reminder-all

Send reminder to all guests (or filtered subset).

Request:

```
{
    "reminder_message": "Reminder: Wedding ceremony tomorrow at 6 PM!",
    "filter_status": "delivered", // Only send to those who received inv
    "filter_invitation_type": null // All invitation types
}
```

Response (200):

```
{  
    "message": "Reminder batch initiated",  
    "event_id": 1,  
    "guests_count": 180,  
    "task_id": "reminder_batch_789012",  
    "status": "processing"  
}
```

POST /api/customer-tab/events/{event_id}/send-thankyou-all

Send thank you message to all guests (or filtered subset).

Request:

```
{  
    "thankyou_message": "Thank you all for making our day special!",  
    "filter_status": "delivered", // Only send to guests who attended (r  
    "filter_invitation_type": null  
}
```

Response (200):

```
{  
    "message": "Thank you batch initiated",  
    "event_id": 1,  
    "guests_count": 180,  
    "task_id": "thankyou_batch_345678",  
    "status": "processing"  
}
```

GET /api/customer-tab/send-status/{task_id}

Check status of a batch send operation (invitations, reminders, or thank you).

Response (200):

```
{  
    "task_id": "send_batch_123456",  
    "task_type": "invitation", // or "reminder" or "thankyou"  
    "status": "completed",  
    "total_guests": 50,  
    "sent": 48,
```

```

    "failed": 2,
    "progress_percentage": 100,
    "started_at": "2024-12-10T15:00:00Z",
    "completed_at": "2024-12-10T15:05:00Z",
    "failed_guests": [
        {
            "guest_id": 45,
            "guest_name": "Neha Sharma",
            "error": "Invalid phone number"
        },
        {
            "guest_id": 67,
            "guest_name": "Vikram Patel",
            "error": "Rate limit exceeded"
        }
    ]
}

```

POST /api/customer-tab/events/{event_id}/generate-report

Generate comprehensive report for event guest management.

Purpose: Generate detailed report with all guest data, send statistics, error analysis, and invitation type breakdown.

Request:

```

{
    "report_type": "detailed", // Options: "summary", "detailed", "error"
    "include_filters": {
        "status": "all", // or specific status
        "invitation_type": "all" // or specific type
    },
    "report_format": "pdf", // Options: "pdf", "excel", "csv"
    "include_sections": {
        "guest_list": true,
        "statistics": true,
        "error_analysis": true,
        "timeline": true,
        "invitation_type_breakdown": true,
        "reminder_thankyou_log": true
    }
}

```

Response (200):

```
{  
    "message": "Report generated successfully",  
    "report_id": "report_123456",  
    "event_id": 1,  
    "event_name": "Raj & Priya Wedding",  
    "report_type": "detailed",  
    "report_format": "pdf",  
    "generated_at": "2024-12-11T15:30:00Z",  
    "download_link": "/api/customer-tab/reports/report_123456/download",  
    "report_summary": {  
        "total_guests": 250,  
        "invitations_sent": 245,  
        "delivered": 230,  
        "failed": 10,  
        "errors": 5,  
        "reminders_sent": 180,  
        "thankyou_sent": 200,  
        "success_rate": 94.0  
    }  
}
```

Report Contents (Detailed Format):

Section 1: Executive Summary

- Event name and details
- Total guests count
- Overall success metrics
- Key highlights

Section 2: Guest List Table

Guest Name	WhatsApp Number	Invitation Type	Status	Sent Date	Delivered Date	Error Details	Reminder Sent	Thank You Sent
Raj Patel	+919998118489	VIP	Delivered	Dec 10, 2:30 PM	Dec 10, 2:31 PM	-	Yes	Yes

Guest Name	WhatsApp Number	Invitation Type	Status	Sent Date	Delivered Date	Error Details	Reminder Sent	Thank You Sent
Amit Kumar	+911234567890	Standard	Error	Dec 10, 2:32 PM	-	Invalid number	No	No

Section 3: Statistics Dashboard

- Total Guests: 250
- Invitations Sent: 245 (98%)
- Successfully Delivered: 230 (94%)
- Failed Deliveries: 10 (4%)
- Errors: 5 (2%)
- Pending: 5 (2%)
- Reminders Sent: 180 (72%)
- Thank You Messages Sent: 200 (80%)

Section 4: Invitation Type Breakdown

Type	Total	Sent	Delivered	Failed	Error	Success Rate
VIP	50	50	48	1	1	96%
Standard	150	145	140	4	1	96.5%
Family	50	50	42	5	3	84%

Section 5: Error Analysis

- Invalid Number: 3 guests
- Blocked by User: 1 guest
- Rate Limit: 1 guest
- Detailed error list with guest names and recommended actions

Section 6: Timeline

- Template uploaded: Dec 1, 2024
- Zones configured: Dec 2, 2024
- Demo approved: Dec 5, 2024
- Batch generation completed: Dec 8, 2024
- First invitation sent: Dec 10, 2024
- Reminders sent: Dec 12, 2024
- Thank you messages sent: Dec 16, 2024
- Event completed: Dec 15, 2024

Section 7: Recommendations

- List of guests with errors and suggested fixes
- Best practices for future events
- Performance metrics

GET /api/customer-tab/reports/{report_id}/download

Download generated report file.

Response: File download (PDF, Excel, or CSV based on report format)

- PDF: Professional formatted report with charts
- Excel: Spreadsheet with multiple sheets for each section
- CSV: Simple guest list with all data columns

POST /api/customer-tab/events/{event_id}/export-guests

Export guest list with current status (quick export, not full report).

Request:

```
{  
    "format": "csv", // or "excel"  
    "include_columns": ["name", "whatsapp_number", "invitation_type", "se  
    "filter_status": "all"  
}
```

Response: CSV or Excel file download with filtered guest data

8. routes/whatsapp.py - WhatsApp Integration (NEW)

Purpose: Send invitations via WhatsApp Cloud API with delivery tracking.

Configuration:

```
# settings.py  
WHATSAPP_API_URL = "https://graph.facebook.com/v18.0/{phone_number_id}/me  
WHATSAPP_API_TOKEN = "your_access_token"  
WHATSAPP_PHONE_NUMBER_ID = "your_phone_number_id"
```

Key Functions:

```
import httpx
from backend.config import settings

async def send_whatsapp_pdf(
    phone: str,
    pdf_path: str,
    message: str = None,
    guest_id: int = None
) -> dict:
    """
    Send PDF invitation via WhatsApp Cloud API

    Args:
        phone: Recipient phone number (international format)
        pdf_path: Local path to PDF file
        message: Custom message text
        guest_id: Guest ID for tracking

    Returns:
        dict with send status and message ID
    """
    # Validate phone number format
    if not phone.startswith('+'):
        phone = '+' + phone

    # Upload PDF to WhatsApp media endpoint first
    async with httpx.AsyncClient() as client:
        # Step 1: Upload media
        with open(pdf_path, 'rb') as pdf_file:
            media_response = await client.post(
                f"https://graph.facebook.com/v18.0/{settings.WHATSAPP_PHC}",
                headers={"Authorization": f"Bearer {settings.WHATSAPP_API_TOKEN}"},
                files={"file": pdf_file},
                data={"messaging_product": "whatsapp"}
            )

        if media_response.status_code != 200:
            raise HTTPException(status_code=500, detail="Failed to upload file")

        media_id = media_response.json()["id"]

        # Step 2: Send message with media
        message_payload = {
            "messaging_product": "whatsapp",
            "to": phone,
            "type": "document",
            "document": {
                "id": media_id
            }
        }

        response = await client.post(
            f"https://graph.facebook.com/v18.0/{settings.WHATSAPP_PHC}/messages",
            headers={"Authorization": f"Bearer {settings.WHATSAPP_API_TOKEN}"},
```

```

        "recipient_type": "individual",
        "to": phone.replace('+', ''),
        "type": "document",
        "document": {
            "id": media_id,
            "caption": message or "Your personalized invitation card",
            "filename": "invitation.pdf"
        }
    }

send_response = await client.post(
    settings.WHATSAPP_API_URL.format(phone_number_id=settings.WHA
    headers={
        "Authorization": f"Bearer {settings.WHATSAPP_API_TOKEN}",
        "Content-Type": "application/json"
    },
    json=message_payload
)

if send_response.status_code != 200:
    error_detail = send_response.json()
    raise HTTPException(
        status_code=send_response.status_code,
        detail=f"WhatsApp send failed: {error_detail}"
    )

response_data = send_response.json()
message_id = response_data["messages"][0]["id"]

# Update guest send status in database
if guest_id:
    with get_cursor() as cursor:
        cursor.execute("""
            UPDATE guests
            SET sent_status = 'sent',
                sent_at = CURRENT_TIMESTAMP,
                whatsapp_message_id = ?
            WHERE id = ?
        """, (message_id, guest_id))

return {
    "status": "sent",
    "message_id": message_id,

```

```
        "phone": phone
    }
```

Webhook Handler for Delivery Status:

```
@router.post("/webhook")
async def whatsapp_webhook(request: Request):
    """
    Handle WhatsApp delivery status webhooks

    Webhook payload example:
    {
        "entry": [
            {
                "changes": [
                    {
                        "value": {
                            "statuses": [
                                {
                                    "id": "wamid.123...",
                                    "status": "delivered",
                                    "timestamp": "1234567890",
                                    "recipient_id": "919998118489"
                                }
                            ]
                        }
                    }
                ]
            }
        }
    }

    payload = await request.json()

    # Verify webhook signature (security)
    # ... verification code ...

    for entry in payload.get("entry", []):
        for change in entry.get("changes", []):
            value = change.get("value", {})

            # Process status updates
            for status in value.get("statuses", []):
                message_id = status["id"]
                status_value = status["status"] # sent, delivered, read,
                timestamp = status["timestamp"]

            # Update guest status in database
            with get_cursor() as cursor:
```

```

        if status_value == "delivered":
            cursor.execute("""
                UPDATE guests
                SET sent_status = 'delivered',
                    delivered_at = ?
                WHERE whatsapp_message_id = ?
            """, (datetime.fromtimestamp(int(timestamp)), mes
        )

    elif status_value == "failed":
        cursor.execute("""
            UPDATE guests
            SET sent_status = 'failed',
                error_message = ?
            WHERE whatsapp_message_id = ?
        """, (status.get("errors", [{}])[0].get("message"
        )

    return {"status": "ok"}

```

Rate Limiting:

```

from fastapi import HTTPException
from collections import defaultdict
from datetime import datetime, timedelta

# Simple in-memory rate limiter
send_history = defaultdict(list)
RATE_LIMIT_PER_HOUR = 100

def check_rate_limit(user_id: int):
    """Check if user has exceeded sending rate limit"""
    now = datetime.utcnow()
    hour_ago = now - timedelta(hours=1)

    # Clean old entries
    send_history[user_id] = [
        ts for ts in send_history[user_id]
        if ts > hour_ago
    ]

    if len(send_history[user_id]) >= RATE_LIMIT_PER_HOUR:
        raise HTTPException(
            status_code=429,
            detail=f"Rate limit exceeded. Maximum {RATE_LIMIT_PER_HOUR} s

```

```
    send_history[user_id].append(now)
```

9. routes/analytics.py - Dashboard & Analytics (ENHANCED)

Purpose: Provide comprehensive statistics for dashboard display.

Key Endpoints:

GET /api/analytics/dashboard

Get dashboard statistics (role-filtered).

Response for Super Admin (200):

```
{
  "role": "sudo_admin",
  "statistics": {
    "total_members": 150,
    "total_customers": 45,
    "total_admins": 5,
    "total_events": 78,
    "cards_sent": 15000,
    "cards_delivered": 13500,
    "delivery_rate_percentage": 90.0,
    "active_events": 12,
    "completed_events": 66,
    "pending_approvals": 3,
    "failed_deliveries": 1500
  },
  "recent_activity": [
    {
      "type": "event_created",
      "message": "New event 'Raj Wedding' created by Admin John",
      "timestamp": "2024-12-10T10:30:00Z"
    },
    {
      "type": "demo_approved",
      "message": "Customer Priya approved demo for event 'Anniversa",
      "timestamp": "2024-12-10T09:15:00Z"
    }
  ],
}
```

```

    "top_admins_by_customers": [
        {"admin_name": "John Doe", "customer_count": 12, "cards_sent": 30},
        {"admin_name": "Jane Smith", "customer_count": 8, "cards_sent": 25}
    ]
}

```

Response for Admin (200):

```

{
    "role": "admin",
    "statistics": {
        "my_customers": 12,
        "my_events": 18,
        "cards_sent": 3000,
        "cards_delivered": 2700,
        "delivery_rate_percentage": 90.0,
        "active_events": 3,
        "completed_events": 15,
        "pending_approvals": 1
    },
    "recent_activity": [
        {
            "type": "cards_sent",
            "message": "50 cards sent for event 'Wedding Ceremony'",
            "timestamp": "2024-12-10T14:00:00Z"
        }
    ]
}

```

GET /api/analytics/events-over-time

Get event creation trends.

Query Parameters:

- `period` : day, week, month, year
- `start_date` : ISO format date
- `end_date` : ISO format date

Response (200):

```
{
    "period": "month",
}
```

```
"data": [
    {"date": "2024-01", "event_count": 5, "cards_sent": 1200},
    {"date": "2024-02", "event_count": 7, "cards_sent": 1800},
    {"date": "2024-03", "event_count": 10, "cards_sent": 2500}
]
}
```

GET /api/analytics/delivery-stats

Get detailed delivery statistics.

Response (200):

```
{
    "total_cards_sent": 15000,
    "delivered": 13500,
    "pending": 500,
    "failed": 1000,
    "delivery_rate": 90.0,
    "average_delivery_time_minutes": 2.5,
    "by_event_type": {
        "wedding": {"sent": 10000, "delivered": 9000, "rate": 90.0},
        "birthday": {"sent": 3000, "delivered": 2700, "rate": 90.0},
        "corporate": {"sent": 2000, "delivered": 1800, "rate": 90.0}
    }
}
```

GET /api/analytics/rsvp-stats

Get RSVP response statistics (if RSVP feature implemented).

Response (200):

```
{
    "total_invitations": 15000,
    "responses": 9000,
    "response_rate": 60.0,
    "attending": 7500,
    "not_attending": 1500,
    "pending": 6000
}
```

10. routes/notifications.py - Notification System (NEW)

Purpose: Manage in-app and email notifications for users.

Database Schema:

```
CREATE TABLE notifications (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL REFERENCES users(id),
    type TEXT CHECK(type IN ('info', 'success', 'warning', 'error')),
    title TEXT NOT NULL,
    message TEXT NOT NULL,
    is_read BOOLEAN DEFAULT 0,
    action_url TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Key Endpoints:

GET /api/notifications

Get user's notifications.

Query Parameters:

- `unread_only` : true/false
- `limit` : Number of notifications

Response (200):

```
{
    "notifications": [
        {
            "id": 1,
            "type": "info",
            "title": "New Customer Assigned",
            "message": "Customer 'Priya Shah' has been assigned to you",
            "is_read": false,
            "action_url": "/customers/10",
            "created_at": "2024-12-10T10:00:00Z"
        },
        {
            "id": 2,
            "type": "success",
            "title": "Customer Assigned",
            "message": "Customer 'Priya Shah' has been assigned to you",
            "is_read": false,
            "action_url": "/customers/10",
            "created_at": "2024-12-10T10:00:00Z"
        }
    ]
}
```

```

        "title": "Demo Approved",
        "message": "Customer approved demo for event 'Wedding Ceremony'",
        "is_read": false,
        "action_url": "/design-studio/1",
        "created_at": "2024-12-09T15:30:00Z"
    }
],
"unread_count": 5
}

```

POST /api/notifications/{notification_id}/mark-read

Mark a notification as read.

POST /api/notifications/mark-all-read

Mark all notifications as read.

Notification Types:

Type	Trigger	Recipient	Example
customer_assigned	Super Admin assigns customer	Admin	"Customer Raj Patel assigned to you"
demo_pending_approval	Admin creates demo	Customer	"Demo card ready for your review"
demo_approved	Customer approves demo	Admin	"Demo approved for Wedding event"
demo_rejected	Customer rejects demo	Admin	"Demo rejected with feedback"
cards_generated	Batch generation complete	Admin	"250 cards generated successfully"
send_complete	WhatsApp batch send done	Admin	"200 cards sent successfully"
delivery_failed	WhatsApp delivery fails	Admin	"20 cards failed to deliver"
event_completed	Event marked complete	Admin, Customer	"Wedding event completed"
password_reset	Password reset requested	User	"Password reset link sent to email"

11. routes/settings.py - Settings Management (NEW)

Purpose: Manage global and user-specific settings.

Database Schema:

```
CREATE TABLE settings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER REFERENCES users(id), -- NULL for global settings
    setting_key TEXT NOT NULL,
    setting_value TEXT NOT NULL,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, setting_key)
);
```

Key Endpoints:

GET /api/settings/global

Get global system settings (Super Admin only).

Response (200):

```
{
    "whatsapp": {
        "api_url": "https://graph.facebook.com/v18.0/...",
        "phone_number_id": "123456789",
        "is_configured": true
    },
    "email": {
        "smtp_host": "smtp.gmail.com",
        "smtp_port": 587,
        "from_email": "noreply@invitewala.com",
        "is_configured": true
    },
    "file_storage": {
        "max_upload_size_mb": 10,
        "auto_delete_after_days": 30
    },
    "rate_limits": {
        "whatsapp_sends_per_hour": 100,
        "pdf_generation_per_batch": 500
    }
}
```

PUT /api/settings/global

Update global settings (Super Admin only).

GET /api/settings/user

Get current user's personal settings.

Response (200):

```
{  
    "theme": "light",  
    "notifications_enabled": true,  
    "email_notifications": true,  
    "language": "en",  
    "timezone": "Asia/Kolkata",  
    "dashboard_view": "cards"  
}
```

PUT /api/settings/user

Update user's personal settings.

12. routes/queries.py - Customer Inquiry Management (NEW)

Purpose: Handle customer inquiries from online/offline forms.

Database Schema:

```
CREATE TABLE queries (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT NOT NULL,  
    email TEXT,  
    phone TEXT NOT NULL,  
    whatsapp_number TEXT,  
    event_type TEXT,  
    event_date DATE,  
    message TEXT,  
    source TEXT CHECK(source IN ('online', 'offline')),  
    status TEXT CHECK(status IN ('pending', 'contacted', 'converted', 'closed')),  
    assigned_admin_id INTEGER REFERENCES users(id),  
    notes TEXT,
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Key Endpoints:

POST /api/queries

Submit a new customer inquiry.

Request:

```
{  
    "name": "Amit Kumar",  
    "email": "amit@example.com",  
    "phone": "+919123456789",  
    "whatsapp_number": "+919123456789",  
    "event_type": "wedding",  
    "event_date": "2025-03-15",  
    "message": "Interested in personalized wedding invitations for 300 guests.",  
    "source": "online"  
}
```

Response (201):

```
{  
    "message": "Inquiry submitted successfully",  
    "query_id": 25,  
    "reference_number": "INQ-25-20241210"  
}
```

GET /api/queries

List all inquiries (Super Admin: all, Admin: assigned).

Response (200):

```
{  
    "queries": [  
        {  
            "id": 25,  
            "name": "Amit Kumar",  
            "email": "amit@example.com",  
            "phone": "+919123456789",  
            "event_type": "wedding",  
            "event_date": "2025-03-15",  
            "message": "Interested in personalized wedding invitations for 300 guests.",  
            "source": "online",  
            "status": "assigned",  
            "assigned_to": "John Doe",  
            "created_at": "2024-12-10T12:00:00Z",  
            "updated_at": "2024-12-10T12:00:00Z"  
        }  
    ]  
}
```

```
        "event_type": "wedding",
        "event_date": "2025-03-15",
        "status": "pending",
        "assigned_admin_id": null,
        "assigned_admin_name": null,
        "created_at": "2024-12-10T09:00:00Z"
    }
],
"total": 1
}
```

POST /api/queries/{query_id}/assign

Assign inquiry to an admin (Super Admin only).

PUT /api/queries/{query_id}/status

Update inquiry status.

POST /api/queries/{query_id}/convert

Convert inquiry to customer.

Request:

```
{
    "first_name": "Amit",
    "last_name": "Kumar",
    "address": "Mumbai, Maharashtra",
    "notes": "Converted from inquiry #25"
}
```

Response (201):

```
{
    "message": "Inquiry converted to customer",
    "customer_id": 50,
    "query_id": 25
}
```

Core Engine Architecture

```
pdf_personalizer/
└── core/
    ├── overlayer.py          # Text placement on PDF
    ├── masking.py            # Background preparation
    ├── generator.py          # Batch processing
    └── conflict_detector.py # Zone overlap detection
```

overlayer.py - Text Overlay Module

Purpose: Place personalized text on PDF pages at defined zones.

Main Function:

```
import fitz  # PyMuPDF
from typing import Dict, List

def overlay_text_on_pdf(
    template_path: str,
    zones: List[Dict],
    text_data: Dict[str, str],
    output_path: str
) -> str:
    """
    Apply text overlays to PDF template
    """

    Apply text overlays to PDF template
```

Args:

```
    template_path: Path to template PDF
    zones: List of zone definitions with styling
    text_data: Dictionary mapping CSV columns to values
    output_path: Where to save personalized PDF
```

Returns:

```
    Path to generated PDF
```

Example zone:

```
{
    "page": 0,
    "x": 100, "y": 200, "width": 300, "height": 50,
    "csv_column": "name",
    "font_family": "Arial",
```

```
"font_size": 24,
"font_color": "#000000",
"alignment": "center",
"mask_mode": "auto_sample"
}
"""

# Open PDF
doc = fitz.open(template_path)

for zone in zones:
    page = doc[zone["page"]]

    # Apply masking first (if needed)
    if zone["mask_mode"] != "overlay":
        apply_mask(page, zone)

    # Get text to insert
    text = text_data.get(zone["csv_column"], "")
    if not text:
        continue

    # Calculate position based on alignment
    rect = fitz.Rect(
        zone["x"],
        zone["y"],
        zone["x"] + zone["width"],
        zone["y"] + zone["height"]
    )

    # Parse color
    color = hex_to_rgb(zone["font_color"])

    # Insert text
    page.insert_textbox(
        rect,
        text,
        fontsize=zone["font_size"],
        fontname=zone["font_family"],
        color=color,
        align=get_alignment_code(zone["alignment"])
    )

# Save modified PDF
doc.save(output_path)
```

```

    doc.close()

    return output_path

def hex_to_rgb(hex_color: str) -> tuple:
    """Convert hex color to RGB tuple (0-1 range)"""
    hex_color = hex_color.lstrip('#')
    return tuple(int(hex_color[i:i+2], 16) / 255 for i in (0, 2, 4))

def get_alignment_code(alignment: str) -> int:
    """Convert alignment string to PyMuPDF code"""

    return {
        "left": 0,
        "center": 1,
        "right": 2,
        "justify": 3
    }.get(alignment, 0)

```

Supported Fonts:

- Arial
- Times New Roman
- Courier New
- Helvetica
- Custom fonts (if installed on server)

Text Styling Options:

- Font size: 8-72 points
 - Color: Any hex color (#RRGGBB)
 - Alignment: left, center, right, justify
 - Bold/Italic: Via font name (e.g., "Arial-Bold")
-

masking.py - Background Masking Module

Purpose: Prepare zones by removing or masking existing content.

Mask Modes:

Mode	Description	Best For	Performance
overlay	No masking, text directly on top	Transparent zones, blank areas	Fastest

Mode	Description	Best For	Performance
auto_sample	Sample corner colors and fill rectangle	Solid color backgrounds	Fast
magic	Remove text objects, preserve textures	Patterned backgrounds	Medium
solid	Fill with custom color	Any background with custom color	Fast

Main Function:

```

import fitz
from PIL import Image, ImageDraw

def apply_mask(
    page: fitz.Page,
    zone: Dict,
    mode: str = "auto_sample",
    custom_color: str = None
) -> None:
    """
    Apply background masking to a zone

    Args:
        page: PyMuPDF page object
        zone: Zone definition dict
        mode: Masking mode
        custom_color: Custom color for 'solid' mode
    """
    rect = fitz.Rect(zone["x"], zone["y"],
                     zone["x"] + zone["width"],
                     zone["y"] + zone["height"])

    if mode == "overlay":
        # No masking needed
        return

    elif mode == "auto_sample":
        # Sample colors from zone corners
        pix = page.get_pixmap(clip=rect)
        img = Image.frombytes("RGB", (pix.width, pix.height), pix.samples)

        # Get corner colors (4 corners)
        corners = [
            img.getpixel((0, 0)),
            img.getpixel((0, img.height - 1)),
            img.getpixel((img.width - 1, 0)),
            img.getpixel((img.width - 1, img.height - 1))
        ]
    
```

```

        img.getpixel((0, 0)),
        img.getpixel((pix.width-1, 0)),
        img.getpixel((0, pix.height-1)),
        img.getpixel((pix.width-1, pix.height-1))
    ]

# Average color
avg_color = tuple(sum(c[i] for c in corners) // 4 for i in range(3))
fill_color = tuple(c / 255 for c in avg_color)

# Draw filled rectangle
page.draw_rect(rect, color=fill_color, fill=fill_color)

elif mode == "magic":
    # Remove text objects in zone
    # This preserves background patterns/images
    for item in page.get_text("dict")["blocks"]:
        if item["type"] == 0:  # Text block
            item_rect = fitz.Rect(item["bbox"])
            if rect.intersects(item_rect):
                # Redact text
                page.add_redact_annot(item_rect, fill=(1, 1, 1))
    page.apply_redactions()

elif mode == "solid":
    # Fill with custom color
    if custom_color:
        fill_color = hex_to_rgb(custom_color)
        page.draw_rect(rect, color=fill_color, fill=fill_color)

```

Mode Selection Guide:

- **Blank/transparent zones:** Use "overlay"
 - **Solid backgrounds:** Use "auto_sample" or "solid"
 - **Textured backgrounds with existing text:** Use "magic"
 - **Specific color needed:** Use "solid" with custom_color
-

generator.py - Batch Generation Module

Purpose: Generate personalized PDFs for all guests efficiently.

Main Function:

```
import os
import csv
from concurrent.futures import ThreadPoolExecutor, as_completed
from typing import List, Dict


def generate_batch(
    template_path: str,
    zones: List[Dict],
    csv_path: str,
    output_dir: str,
    max_workers: int = 4
) -> Dict:
    """
    Generate personalized PDFs for all CSV rows
    """

    Args:
        template_path: Path to template PDF
        zones: Zone definitions
        csv_path: Path to CSV file
        output_dir: Output directory
        max_workers: Number of parallel workers

    Returns:
        Generation results dict
    """

    # Create output directory
    os.makedirs(output_dir, exist_ok=True)

    # Read CSV
    with open(csv_path, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)
        csv_data = list(reader)

    results = {
        "total": len(csv_data),
        "success": 0,
        "failed": 0,
        "outputs": [],
        "errors": []
    }

    # Parallel generation
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        futures = []
```

```

        for idx, row_data in enumerate(csv_data):
            output_path = os.path.join(output_dir, f"{idx}_card.pdf")

            future = executor.submit(
                overlay_text_on_pdf,
                template_path,
                zones,
                row_data,
                output_path
            )
            futures.append((future, idx, row_data, output_path))

    # Collect results
    for future, idx, row_data, output_path in futures:
        try:
            result_path = future.result()
            results["success"] += 1
            results["outputs"].append({
                "index": idx,
                "path": result_path,
                "guest_name": row_data.get("name", "Unknown")
            })
        except Exception as e:
            results["failed"] += 1
            results["errors"].append({
                "index": idx,
                "guest_name": row_data.get("name", "Unknown"),
                "error": str(e)
            })

    return results

```

Performance Optimization:

- **Parallel Processing:** Uses ThreadPoolExecutor for concurrent PDF generation
- **Typical Speed:** 50-100 PDFs per minute (depends on PDF complexity)
- **Memory Management:** Processes in batches if list is very large (>1000)
- **Error Handling:** Continues processing even if some PDFs fail

Large Batch Handling:

```

def generate_large_batch(template_path, zones, csv_path, output_dir, batch_size):
    """Handle very large guest lists by processing in chunks"""

```

```

with open(csv_path, 'r', encoding='utf-8') as f:
    reader = csv.DictReader(f)

    batch = []
    batch_num = 0

    for row in reader:
        batch.append(row)

        if len(batch) >= batch_size:
            # Process batch
            process_batch(template_path, zones, batch, output_dir, batch_num)
            batch = []
            batch_num += 1

    # Process remaining
    if batch:
        process_batch(template_path, zones, batch, output_dir, batch_num)

```

conflict_detector.py - Zone Overlap Detection (NEW)

Purpose: Detect and warn about overlapping zones that could cause rendering issues.

Main Function:

```

from typing import List, Dict, Tuple

def detect_zone_conflicts(zones: List[Dict]) -> List[Dict]:
    """
    Detect overlapping zones

```

Args:

 zones: List of zone definitions

Returns:

 List of conflict reports

Example conflict:

```
{
    "zone1_id": "zone1",
    "zone2_id": "zone2",
    "overlap_area": 1500, # square pixels
    "overlap_percentage": 50.0,
```

```

        "severity": "high",
        "recommendation": "Adjust zone2 position"
    }
"""

conflicts = []

for i, zone1 in enumerate(zones):
    for j, zone2 in enumerate(zones[i+1:], start=i+1):
        # Only check zones on same page
        if zone1["page"] != zone2["page"]:
            continue

        # Calculate intersection
        intersection = calculate_intersection(zone1, zone2)

        if intersection > 0:
            zone1_area = zone1["width"] * zone1["height"]
            zone2_area = zone2["width"] * zone2["height"]

            # Calculate overlap percentage
            overlap_pct = (intersection / min(zone1_area, zone2_area))

            conflicts.append({
                "zone1_id": zone1["id"],
                "zone2_id": zone2["id"],
                "overlap_area": intersection,
                "overlap_percentage": overlap_pct,
                "severity": get_severity(overlap_pct),
                "recommendation": f"Adjust {zone2['id']} position to"
            })

return conflicts

def calculate_intersection(zone1: Dict, zone2: Dict) -> float:
    """Calculate intersection area between two zones"""
    x1 = max(zone1["x"], zone2["x"])
    y1 = max(zone1["y"], zone2["y"])
    x2 = min(zone1["x"] + zone1["width"], zone2["x"] + zone2["width"])
    y2 = min(zone1["y"] + zone1["height"], zone2["y"] + zone2["height"])

    if x2 > x1 and y2 > y1:
        return (x2 - x1) * (y2 - y1)
    return 0

```

```

def get_severity(overlap_percentage: float) -> str:
    """Determine conflict severity"""
    if overlap_percentage > 50:
        return "high"
    elif overlap_percentage > 20:
        return "medium"
    else:
        return "low"

```

UI Integration:

```

// Frontend warns user when conflicts detected
if (response.conflicts_detected) {
    showWarning({
        title: "Zone Conflicts Detected",
        message: `${response.conflicts.length} zones are overlapping`,
        conflicts: response.conflicts,
        actions: ["Adjust Zones", "Continue Anyway"]
    });
}

```

Database Schema

Complete Database Schema

```

-- =====
-- USERS TABLE
-- =====

CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    name TEXT NOT NULL,
    mobile TEXT NOT NULL,
    role TEXT CHECK(role IN ('sudo_admin', 'admin', 'customer')) NOT NULL,
    is_verified BOOLEAN DEFAULT 0,
    verification_token TEXT,
    reset_token TEXT,
    reset_token_expires TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
)

```

```

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
last_login TIMESTAMP
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);

-- =====
-- CUSTOMERS TABLE
-- =====

CREATE TABLE customers (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    first_name TEXT NOT NULL,
    last_name TEXT,
    email TEXT,
    phone TEXT,
    whatsapp_number TEXT,
    address TEXT,
    notes TEXT,
    assigned_admin_id INTEGER REFERENCES users(id),
    priority TEXT CHECK(priority IN ('low', 'medium', 'high')) DEFAULT 'n
    status TEXT CHECK(status IN ('active', 'inactive', 'archived')) DEFAU
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_customers_assigned_admin ON customers(assigned_admin_id)
CREATE INDEX idx_customers_phone ON customers(phone);

-- =====
-- EVENTS TABLE
-- =====

CREATE TABLE events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL REFERENCES customers(id) ON DELETE CASCADE,
    event_name TEXT NOT NULL,
    event_type TEXT CHECK(event_type IN ('wedding', 'birthday', 'annivers
    event_date DATE,
    venue TEXT,
    notes TEXT,
    status TEXT CHECK(status IN ('pending', 'active', 'completed', 'cance

    -- File paths
    template_pdf_path TEXT,

```

```

guest_csv_path TEXT,
zones_config TEXT, -- JSON string of zones

-- Demo workflow
demo_approved BOOLEAN DEFAULT 0,
demo_pdf_path TEXT,
demo_rejection_reason TEXT,

-- Statistics
total_guests INTEGER DEFAULT 0,
cards_generated INTEGER DEFAULT 0,
cards_sent INTEGER DEFAULT 0,
cards_delivered INTEGER DEFAULT 0,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
completed_at TIMESTAMP
);

CREATE INDEX idx_events_customer ON events(customer_id);
CREATE INDEX idx_events_status ON events(status);
CREATE INDEX idx_events_date ON events(event_date);

-- =====
-- GUESTS TABLE
-- =====

CREATE TABLE guests (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_id INTEGER NOT NULL REFERENCES events(id) ON DELETE CASCADE,

    -- Guest information
    name TEXT NOT NULL,
    phone TEXT,
    whatsapp_number TEXT, -- Dedicated WhatsApp number (may differ from
    invitation_type TEXT,
    csv_row_index INTEGER,
    csv_data TEXT, -- JSON string of full CSV row

    -- PDF information
    pdf_path TEXT,
    pdf_name TEXT, -- Human-readable filename (e.g., "Raj_Patel_VIP_Invi

    -- Send status tracking
    sent_status TEXT CHECK(sent_status IN ('pending', 'sent', 'delivered'))

```

```

sent_at TIMESTAMP,
delivered_at TIMESTAMP,

-- WhatsApp tracking
whatsapp_message_id TEXT,
custom_message TEXT,

-- Error tracking with detailed categorization
error_code TEXT,          -- WhatsApp API error code (e.g., "131026")
error_message TEXT,        -- Detailed error description from API
error_type TEXT,          -- Categorized error type (invalid_number, bloc

-- Additional messaging features
reminder_sent BOOLEAN DEFAULT 0,
reminder_sent_at TIMESTAMP,
thankyou_sent BOOLEAN DEFAULT 0,
thankyou_sent_at TIMESTAMP,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

```

CREATE INDEX idx_guests_event ON guests(event_id);
CREATE INDEX idx_guests_status ON guests(sent_status);
CREATE INDEX idx_guests_phone ON guests(phone);
CREATE INDEX idx_guests_whatsapp ON guests(whatsapp_number);
CREATE INDEX idx_guests_invitation_type ON guests(invitation_type);
CREATE INDEX idx_guests_error_type ON guests(error_type);

```

```

-- =====
-- WHATSAPP LOGS TABLE
-- =====

CREATE TABLE whatsapp_logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    guest_id INTEGER REFERENCES guests(id),
    event_id INTEGER REFERENCES events(id),
    phone TEXT NOT NULL,
    pdf_path TEXT,
    message_text TEXT,
    whatsapp_message_id TEXT,
    status TEXT CHECK(status IN ('pending', 'sent', 'delivered', 'read',
    error_code TEXT,
    error_message TEXT,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    delivered_at TIMESTAMP,
```

```

    read_at TIMESTAMP
);

CREATE INDEX idx_whatsapp_logs_guest ON whatsapp_logs(guest_id);
CREATE INDEX idx_whatsapp_logs_status ON whatsapp_logs(status);

-- =====
-- NOTIFICATIONS TABLE
-- =====

CREATE TABLE notifications (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    type TEXT CHECK(type IN ('info', 'success', 'warning', 'error')) NOT
    title TEXT NOT NULL,
    message TEXT NOT NULL,
    is_read BOOLEAN DEFAULT 0,
    action_url TEXT,
    related_entity_type TEXT, -- customer, event, guest, etc.
    related_entity_id INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_notifications_user ON notifications(user_id);
CREATE INDEX idx_notifications_read ON notifications(is_read);

-- =====
-- QUERIES (INQUIRIES) TABLE
-- =====

CREATE TABLE queries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT,
    phone TEXT NOT NULL,
    whatsapp_number TEXT,
    event_type TEXT,
    event_date DATE,
    message TEXT,
    source TEXT CHECK(source IN ('online', 'offline')) NOT NULL,
    status TEXT CHECK(status IN ('pending', 'contacted', 'converted', 'cl
    assigned_admin_id INTEGER REFERENCES users(id),
    converted_customer_id INTEGER REFERENCES customers(id),
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)

```

```

) ;

CREATE INDEX idx_queries_status ON queries(status);
CREATE INDEX idx_queries_assigned_admin ON queries(assigned_admin_id);

-- =====
-- SETTINGS TABLE
-- =====

CREATE TABLE settings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER REFERENCES users(id), -- NULL for global settings
    setting_key TEXT NOT NULL,
    setting_value TEXT NOT NULL,
    setting_type TEXT CHECK(setting_type IN ('string', 'number', 'boolean')),
    is_global BOOLEAN DEFAULT 0,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, setting_key)
);

CREATE INDEX idx_settings_user ON settings(user_id);
CREATE INDEX idx_settings_key ON settings(setting_key);

-- =====
-- TASKS TABLE (Optional - for admin task management)
-- =====

CREATE TABLE tasks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    description TEXT,
    status TEXT CHECK(status IN ('pending', 'in_progress', 'completed', 'cancelled')),
    priority TEXT CHECK(priority IN ('low', 'medium', 'high')) DEFAULT 'medium',
    assigned_to INTEGER REFERENCES users(id),
    related_event_id INTEGER REFERENCES events(id),
    due_date DATE,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_tasks_assigned_to ON tasks(assigned_to);
CREATE INDEX idx_tasks_status ON tasks(status);

-- =====
-- AUDIT LOGS TABLE (Optional - for compliance)

```

```
-- =====
CREATE TABLE audit_logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER REFERENCES users(id),
    action TEXT NOT NULL,
    entity_type TEXT NOT NULL,
    entity_id INTEGER,
    details TEXT, -- JSON string
    ip_address TEXT,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_audit_logs_user ON audit_logs(user_id);
CREATE INDEX idx_audit_logs_entity ON audit_logs(entity_type, entity_id);
CREATE INDEX idx_audit_logs_created_at ON audit_logs(created_at);
```

Database Initialization

```
# database/database.py
import sqlite3
from contextlib import contextmanager

DATABASE_PATH = "invitewala.db"

def init_db():
    """Initialize database with all tables"""
    conn = sqlite3.connect(DATABASE_PATH)
    cursor = conn.cursor()

    # Read and execute schema from file
    with open('database/schema.sql', 'r') as f:
        schema = f.read()
        cursor.executescript(schema)

    # Create default super admin if not exists
    cursor.execute("SELECT COUNT(*) FROM users WHERE role = 'sudo_admin'")
    if cursor.fetchone()[0] == 0:
        from backend.auth.auth import hash_password
        cursor.execute("""
            INSERT INTO users (email, password_hash, name, mobile, role,
            VALUES (?, ?, ?, ?, ?, ?)
        
```

```

""" , (
    "admin@invitewala.com",
    hash_password("admin123"),
    "Super Admin",
    "+919999999999",
    "sudo_admin",
    1
))

conn.commit()
conn.close()

@contextmanager
def get_cursor():
    """Context manager for database operations"""
    conn = sqlite3.connect(DATABASE_PATH)
    conn.row_factory = sqlite3.Row
    cursor = conn.cursor()
    try:
        yield cursor
        conn.commit()
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        conn.close()

```

API Reference

API Documentation

Complete API documentation is auto-generated by FastAPI and available at:

- Swagger UI: <http://localhost:5234/api/docs>
- ReDoc: <http://localhost:5234/api/redoc>

Authentication

All endpoints (except /api/auth/login and /api/auth/register) require authentication.

Include token in request headers:

Authorization: Bearer {access_token}

HTTP Status Codes

Code	Meaning	Usage
200	OK	Successful GET, PUT, DELETE
201	Created	Successful POST (resource created)
204	No Content	Successful DELETE (no response body)
400	Bad Request	Invalid input data
401	Unauthorized	Missing or invalid token
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
409	Conflict	Duplicate resource
422	Unprocessable Entity	Validation error
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server error

Error Response Format

```
{  
    "detail": "Error message describing what went wrong",  
    "error_code": "VALIDATION_ERROR",  
    "field": "email", // (optional) specific field with error  
    "timestamp": "2024-12-10T15:30:00Z"  
}
```

Authentication & Authorization

JWT Token Flow

1. User Login

```
↳ POST /api/auth/login  
    ↳ Validate credentials  
        ↳ Generate JWT token
```

↳ Return token + user info

2. Authenticated Request

↳ Include "Authorization: Bearer <token>" header

↳ Backend validates token

↳ Extract user info from token

↳ Check role permissions

↳ Process request or return 403

Token Structure

```
{  
  "user_id": 2,  
  "email": "admin@invitewala.com",  
  "role": "admin",  
  "exp": 1702483200, // Expiration timestamp  
  "iat": 1702396800 // Issued at timestamp  
}
```

Role Permissions Matrix

Endpoint	Super Admin	Admin	Customer
User Management			
GET /api/users	✓	✗	✗
POST /api/users	✓	✗	✗
PUT /api/users/{id}	✓	⚠ Own only	⚠ Own only
DELETE /api/users/{id}	✓	✗	✗
Customer Management			
GET /api/customers	✓ All	✓ Assigned	✗
POST /api/customers	✓	✓	✗
PUT /api/customers/{id}	✓	✓ Assigned	✗
POST /api/customers/{id}/assign	✓	✗	✗
Event Management			
GET /api/events	✓ All	✓ Own customers	⚠ Own

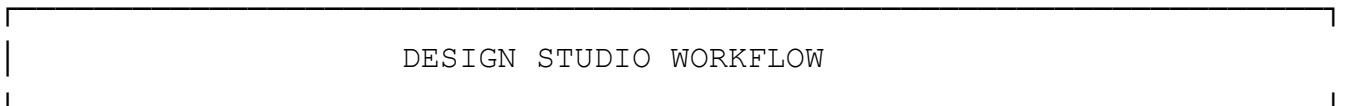
Endpoint	Super Admin	Admin	Customer
POST /api/events	✓	✓	✗
PUT /api/events/{id}	✓	✓ Own customers	✗
DELETE /api/events/{id}	✓	✓ Own customers	✗
Design Studio			
All design endpoints	✓	✓	✗
Customer Tab			
All customer tab endpoints	✓	✓ Assigned	✗
WhatsApp			
Send endpoints	✓	✓	✗
Analytics			
GET /api/analytics/dashboard	✓ Global	✓ Filtered	⚠ Own
Settings			
GET /api/settings/global	✓	✗	✗
PUT /api/settings/global	✓	✗	✗
GET /api/settings/user	✓	✓	✓

Legend:

- ✓ Full access
- ⚠ Limited access (own data only)
- ✗ No access (403 Forbidden)

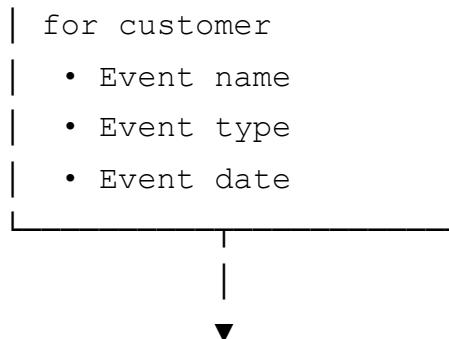
Design Studio Workflow

Complete Workflow Diagram

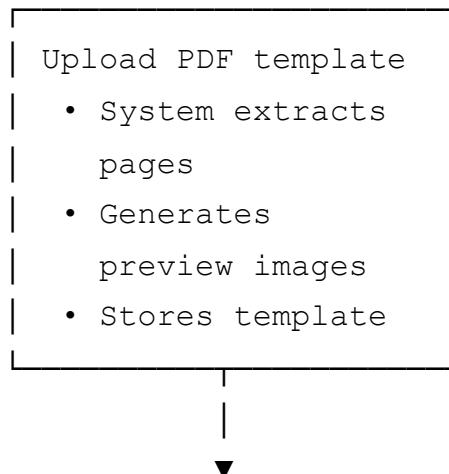


Step 1: CREATE EVENT

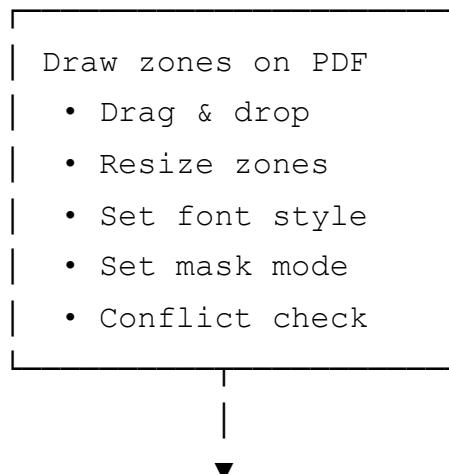




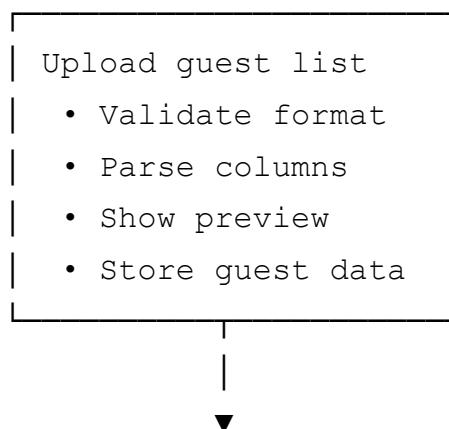
Step 2: UPLOAD TEMPLATE



Step 3: DEFINE ZONES

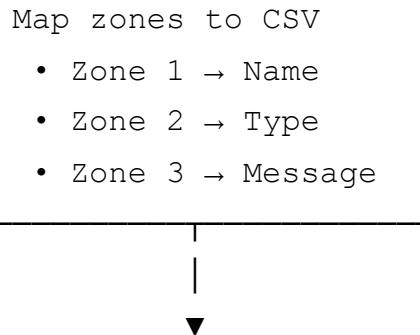


Step 4: UPLOAD CSV

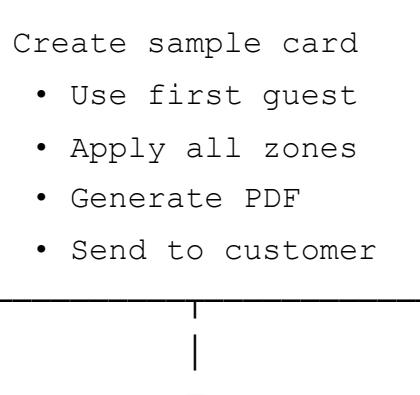


Step 5: MAP COLUMNS

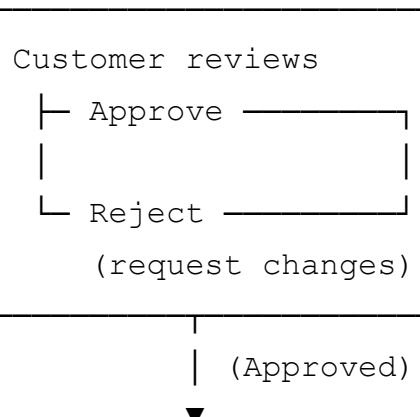




Step 6: GENERATE DEMO

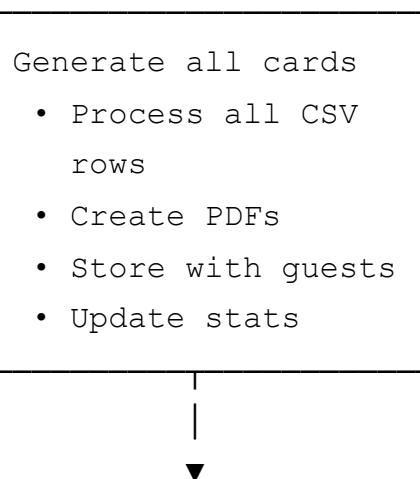


Step 7: DEMO APPROVAL



(Approved)

Step 8: BATCH GENERATION



Step 9: CUSTOMER TAB



- View all guests
- Select guests
- Download PDFs
- Prepare to send



Step 10: SEND VIA WHATSAPP

- Distribute cards
 - Send selected
 - Send all
 - Track delivery
 - Prevent duplicates



Step 11: TRACK & COMPLETE

- Monitor & finalize
 - View statistics
 - Handle failures
 - Complete event
 - Archive/cleanup

Zone Drawing Interface

Coordinate System:

- PDF uses **Points** (72 DPI): 1 inch = 72 points
- Preview images use **Pixels** (150 DPI): 1 inch = 150 pixels
- **Conversion:** `pdf_points = pixel_coords * (72/150)`

Example Zone Drawing:

```
// Frontend: User draws zone on 150 DPI preview image
const zone = {
  x: 150,          // pixels
  y: 300,          // pixels
  width: 450,      // pixels
  height: 75       // pixels
```

```
};

// Convert to PDF coordinates before saving
const pdfZone = {
  x: zone.x * (72/150),           // = 72 points
  y: zone.y * (72/150),           // = 144 points
  width: zone.width * (72/150),   // = 216 points
  height: zone.height * (72/150) // = 36 points
};
```

CSV Format Requirements

Required Columns: At minimum, `name` and `phone` columns are recommended.

Example CSV:

```
name,phone,invitation_type,table_number,special_message
Raj Patel,+919998118489,VIP,A1,"We're so excited to celebrate with you!"
Priya Shah,+919876543210,Standard,B5,
Amit Kumar,+919123456789,VIP,A2,"Your presence means the world to us"
```

Validation Rules:

- UTF-8 encoding required
- Maximum 10,000 rows
- Phone numbers should be in international format (+91XXXXXXXXXX)
- Empty cells are allowed (will result in blank text in that zone)

Customer Management Module

Customer Lifecycle

CUSTOMER LIFECYCLE

1. INQUIRY
 - ├ Online form submission
 - ├ Offline form entry
 - └ Status: "pending"



2. ASSIGNMENT

- └ Super Admin assigns to Admin
 - └ Admin receives notification
- |
- ▼

3. CONTACT & CONVERSION

- └ Admin contacts customer
 - └ Discusses requirements
 - └ Converts inquiry to customer
- |
- ▼

4. CUSTOMER PROFILE CREATION

- └ Create customer record
 - └ Store contact details
 - └ Set priority level
- |
- ▼

5. EVENT CREATION

- └ Admin creates event for customer
 - └ Event details entered
 - └ Status: "pending"
- |
- ▼

6. DESIGN & GENERATION

- └ Design Studio workflow
 - └ Demo approval
 - └ Batch generation
- |
- ▼

7. DISTRIBUTION

- └ Send via WhatsApp
 - └ Track delivery
 - └ Handle errors
- |
- ▼

8. EVENT COMPLETION

- └ Mark event as "completed"
- └ Archive data

```
├─ cleanup files
└─ Customer remains for future events
```

Customer Assignment Rules

Assignment Flow:

1. Customer inquiry received
2. Super Admin reviews inquiry
3. Super Admin selects appropriate Admin based on:
 - Admin workload (number of assigned customers)
 - Admin expertise/specialization
 - Geographic location
4. Admin receives notification
5. Customer appears in Admin's customer list

Re-assignment:

- Super Admin can reassign customer to different Admin
- All event data moves with customer
- Previous Admin loses access to customer

Customer Tab Interface & Features

Overview

The Customer Tab is the central interface for managing guest lists, downloading PDFs, sending invitations, and tracking delivery status. It provides comprehensive guest management with advanced filtering, bulk operations, and detailed reporting capabilities.

Guest Table Structure

The guest table displays all guests for an event in a comprehensive, interactive table format with the following columns:

Column Definitions

Column	Width	Description	Features
--------	-------	-------------	----------

Column	Width	Description	Features
Checkbox	40px	Guest selection	<ul style="list-style-type: none"> Select individual guests Used for bulk operations Visual indication when selected
Guest Name	200px	Full name from CSV	<ul style="list-style-type: none"> Sortable Searchable Displays full name from CSV data
WhatsApp Number	150px	Contact number	<ul style="list-style-type: none"> Formatted display (+91-9998-118489) Click to copy Validation indicator Shows if different from phone number
PDF File	250px	Generated PDF link	<ul style="list-style-type: none"> Clickable filename Click to instantly download Shows human-readable name Example: "Raj_Patel_VIP_Invitation.pdf" File size indicator
Invitation Type	120px	Type from CSV	<ul style="list-style-type: none"> Color-coded badges VIP: Gold badge Standard: Blue badge Family: Green badge Custom types supported Filterable
Action Status	180px	Send status & errors	<ul style="list-style-type: none"> Icon-based display Pending: ⏳ Yellow Sent: 📧 Blue Delivered: ✅ Green Failed: ❌ Red Error: ⚠️ Red Shows timestamp Hover for error details Click for full error message
Download	100px	Individual download	<ul style="list-style-type: none"> Download button icon Instant PDF download Shows file size Disabled if PDF not generated
Resend	100px	Resend invitation	<ul style="list-style-type: none"> Only visible for failed/error Opens resend dialog Option to update phone number Tracks resend attempts

Table Header Actions

Located above the table:

1. Select All Checkbox

- Selects/deselects all guests in current view
- Smart selection (only selectable guests)
- Shows count: "250 guests selected"

2. Filter Dropdown

- Multiple filter options (detailed below)
- Shows active filter count
- Reset filters button

3. Search Bar

- Search by name or phone number
- Real-time filtering
- Clear search button

4. Bulk Actions Toolbar (appears when guests selected)

- Download Selected PDFs (ZIP)
- Send Selected
- Send Reminder
- Send Thank You
- Export Selected

5. Quick Stats Bar

- Total: 250 guests
 - Pending: 50 | Sent: 150 | Delivered: 180
 - Failed: 15 | Error: 5
 - Color-coded indicators
-

Advanced Filtering System

Filter Dropdown Interface

Location: Top-right of guest table

Filter Categories:

1. Status Filter (Primary)

Dropdown with radio buttons:

- **All Guests** (250) - Default view
- **Pending** (50) - Not yet sent

-  **Sent** (150) - Sent but not confirmed delivered
-  **Delivered** (180) - Successfully delivered
-  **Failed** (15) - Failed to send
-  **Error** (5) - Encountered errors

Visual Indication: Each option shows count in parentheses and color-coded dot

2. Invitation Type Filter (Secondary)

Dropdown with checkboxes (multi-select):

- **VIP** (50)
- **Standard** (150)
- **Family** (50)
- **[Custom Types from CSV]**

Behavior: Can select multiple types simultaneously

3. Error Type Filter (Tertiary - only shown if errors exist)

Dropdown for error category:

-  **All Errors** (5)
-  **Invalid Number** (3)
-  **Blocked by User** (1)
-  **Rate Limited** (1)
-  **Network Error** (0)
-  **Media Upload Failed** (0)

4. Additional Filters

- **Reminder Sent:** Yes / No / All
- **Thank You Sent:** Yes / No / All
- **Date Range:** Sent between [date] and [date]

Filter Behavior

Combining Filters:

- Filters work with AND logic
- Example: Status=Error AND Type=VIP shows only VIP guests with errors
- Active filters displayed as chips below search bar
- Each chip has X button to remove

Filter Persistence:

- Filters saved in session
- Restored when returning to page
- Reset all filters button available

URL Parameters:

- Filters reflected in URL for sharing
 - Example: `/customer-tab/1?status=error&type=VIP`
-

Action Status Details

Status Display System

Each guest row shows detailed status information in the Action Status column:

Pending Status



Pending

- **Icon:** Hourglass
- **Color:** Yellow (#FCD34D)
- **Meaning:** PDF generated, not yet sent
- **No error information**

Sent Status



Sent

Dec 10, 2:30 PM

- **Icon:** Envelope
- **Color:** Blue (#3B82F6)
- **Meaning:** Sent to WhatsApp, awaiting delivery confirmation
- **Shows:** Timestamp

Delivered Status

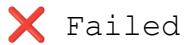


Delivered

Dec 10, 2:31 PM

- **Icon:** Green checkmark
- **Color:** Green (#10B981)
- **Meaning:** Successfully delivered to recipient
- **Shows:** Delivery timestamp
- **Additional:** Read receipt if available

Failed Status



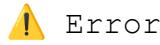
Failed

Rate Limit

Hover for details

- **Icon:** Red X
- **Color:** Red (#EF4444)
- **Meaning:** Send attempt failed
- **Shows:** Brief error reason
- **Hover:** Shows full error message
- **Click:** Opens error detail modal

Error Status



Invalid Number

Click for details

- **Icon:** Warning triangle
- **Color:** Red (#DC2626)
- **Meaning:** Error encountered during send
- **Shows:** Error type
- **Hover tooltip:** Shows error code and message
- **Click:** Opens detailed error modal with:
 - Error code (e.g., 131026)
 - Full error message
 - Suggested action
 - Option to update phone number
 - Resend button

Error Type Categories & Solutions

Error Type	Icon	Description	Suggested Action
invalid_number	🚫	Phone number is invalid or not registered on WhatsApp	Update phone number and resend
blocked	🚫	User has blocked the business number	Contact guest through alternate channel
rate_limit	⌚	Rate limit exceeded, too many messages	Wait and retry later (automatic)
network_error	🌐	Network or connectivity issue	Retry sending

Error Type	Icon	Description	Suggested Action
media_upload_failed		PDF upload to WhatsApp failed	Check PDF size (<16MB), retry
unknown		Unknown error	Check error message, contact support

Button Actions

Individual Guest Actions

Download Button:

- Appearance:** Download icon ()
- Click behavior:** Instant PDF download
- Filename:** `{name}_{type}_Invitation.pdf`
- Disabled state:** Gray when PDF not available

Resend Button:

- Visibility:** Only shown for `failed` or `error` status
- Appearance:** Retry icon () with text "Resend"
- Click behavior:** Opens resend dialog
- Dialog contents:**
 - Current phone number (editable)
 - Custom message textarea
 - Previous error information
 - Number of previous attempts
 - "Resend" and "Cancel" buttons

Bulk Action Buttons

Located in toolbar (appears when guests selected):

1. Download Selected PDFs

Download Selected (5)

- Downloads ZIP file containing selected guests' PDFs
- Shows progress bar during ZIP creation
- Filename: `Event_{name}_Selected_{count}_Invitations.zip`

2. Send Selected



- Opens bulk send dialog
- Custom message textarea
- Preview: List of selected guests
- Warning if any have already been sent
- Confirmation required
- Progress indicator during sending

3. Send Reminder



- Opens reminder dialog
- **Purpose:** Follow-up message to guests
- Custom reminder message textarea
- Checkboxes:
 - Only to delivered guests
 - Only to guests who haven't responded
- Preview selected guests
- Send button with confirmation

Default reminder message template:

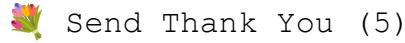
Hello {name},

This is a gentle reminder about our {event_type} on {date}.

We're looking forward to seeing you!

Best regards,
{customer_name}

4. Send Thank You



- Opens thank you dialog
- **Purpose:** Post-event appreciation message
- Custom thank you message textarea
- Preview selected guests
- Send button with confirmation

Default thank you message template:

Dear {name},

Thank you so much for being part of our special day!

Your presence made it truly memorable.

With gratitude,

{customer_name}

5. Export Selected



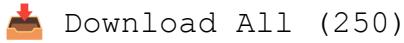
Export (5)

- Export dropdown:
 - CSV format
 - Excel format
 - PDF list
- Includes all guest data columns
- Instant download

Event-Level Actions

Located at top-right of page:

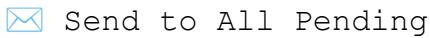
Download All PDFs



Download All (250)

- Downloads ZIP with all guests' PDFs
- Large file warning if >100MB
- Background processing for large batches
- Download link via notification when ready

Send to All



Send to All Pending

- Opens send all dialog
- Shows count of pending guests
- Filter options:
 - Only pending guests
 - Resend to failed guests
 - Filter by invitation type
- Custom message

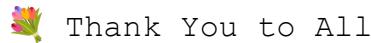
- Batch size configuration (for rate limiting)
- Confirmation with summary

Send Reminder to All



- Send reminder to all delivered guests
- Filter by invitation type
- Custom message
- Confirmation required

Send Thank You to All



- Send thank you to all delivered guests
- Filter by invitation type
- Custom message
- Confirmation required

Generate Report



- Opens report configuration dialog
- Report type selection
- Filters to apply
- Format selection (PDF/Excel/CSV)
- Generate button
- Download link when ready

Report Generation System

Report Configuration Dialog

Trigger: Click "Generate Report" button in Customer Tab

Dialog Contents:

Step 1: Report Type Selection

- **Summary Report:** Key metrics and statistics only
- **Detailed Report:** Full guest list with all data

- **Errors Only Report:** Only guests with errors
- **Invitation Type Report:** Breakdown by invitation type
- **Timeline Report:** Chronological view of all activities

Step 2: Apply Filters

- Use current filters: Yes / No
- If No: Select custom filters
 - Status filter
 - Invitation type filter
 - Date range
 - Error type

Step 3: Report Format

- **PDF:** Professional formatted report with charts
- **Excel:** Spreadsheet with multiple sheets
- **CSV:** Simple data export

Step 4: Include Sections (checkboxes)

- **Guest List Table:** Complete guest data
- **Statistics Dashboard:** Key metrics
- **Error Analysis:** Detailed error breakdown
- **Timeline:** Event timeline
- **Invitation Type Breakdown:** Stats by type
- **Reminder & Thank You Log:** Message history
- **Charts & Graphs:** Visual representations

Step 5: Generate

- **Generate Report** button
- Processing indicator
- Estimated time display
- Download automatically when ready

Detailed Report Contents

PDF Report Structure (Detailed Format)

Cover Page:

- Event name and logo

- Event date and venue
- Report generation date
- Event status badge
- Admin name

Page 1: Executive Summary

EXECUTIVE SUMMARY	
Event:	Raj & Priya Wedding
Date:	December 15, 2024
Total Guests:	250
KEY METRICS	
✓	Invitations Sent: 245 (98%)
✓	Delivered: 230 (94%)
⚠	Failed: 10 (4%)
✗	Errors: 5 (2%)
ADDITIONAL MESSAGING	
🔔	Reminders Sent: 180 (72%)
✉️	Thank You Sent: 200 (80%)
OVERALL SUCCESS RATE: 94%	

Page 2-N: Guest List Table

Professional table with all columns:

- Row number
- Guest Name
- WhatsApp Number
- Invitation Type (color-coded)
- PDF Filename
- Status (icon + text)
- Sent Date & Time
- Delivered Date & Time
- Error Details (if any)
- Reminder Sent (Yes/No)
- Thank You Sent (Yes/No)

Subsequent Pages:

Statistics Dashboard (with charts):

- Pie chart: Status distribution
- Bar chart: Invitation types
- Line chart: Sends over time
- Donut chart: Success vs failures

Invitation Type Breakdown Table:

Type	Total	Sent	Delivered	Failed	Error	Success Rate	Avg. Delivery Time
VIP	50	50	48	1	1	96%	1.2 min
Standard	150	145	140	4	1	96.5%	1.5 min
Family	50	50	42	5	3	84%	2.1 min

Error Analysis Section:

ERROR BREAKDOWN

Total Errors: 5

By Error Type:

- Invalid Number: 3 guests (60%)
 - Amit Kumar: +911234567890
 - Neha Sharma: +919876543211
 - Vikram Patel: +919123456780

Recommendation: Verify and update phone numbers

- Blocked by User: 1 guest (20%)
 - Priya Singh: +919998118490

Recommendation: Contact through alternate channel

- Rate Limited: 1 guest (20%)
 - Raj Mehta: +919876543299

Recommendation: Automatically retried, now delivered

Timeline Section:

EVENT TIMELINE

- ✓ Dec 1, 2024 - Event created
- ✓ Dec 1, 2024 - Template uploaded
- ✓ Dec 2, 2024 - Zones configured
- ✓ Dec 2, 2024 - CSV uploaded (250 guests)
- ✓ Dec 5, 2024 - Demo generated
- ✓ Dec 5, 2024 - Demo approved by customer
- ✓ Dec 8, 2024 - Batch generation started
- ✓ Dec 8, 2024 - Batch generation completed (250 PDFs)
- ✓ Dec 10, 2024 - First invitation sent (2:30 PM)
- ✓ Dec 10, 2024 - Bulk send completed (3:15 PM)
 - Dec 10, 2024 - 5 errors encountered
- ✓ Dec 12, 2024 - Reminders sent to 180 guests
- ✓ Dec 15, 2024 - Event day
- ✓ Dec 16, 2024 - Thank you messages sent
- ✓ Dec 17, 2024 - Event marked completed

Recommendations Section:

- List of guests requiring attention
- Suggested actions for errors
- Best practices for future events
- Performance improvement suggestions

Footer: Page numbers, generation timestamp, confidentiality notice

Excel Report Structure

Multiple worksheets:

1. Summary Sheet

- Key metrics
- Charts(embedded)
- Quick stats

2. Guest List Sheet

- Complete data table
- Sortable columns
- Filtered views
- Conditional formatting (status colors)

3. Errors Sheet

- Only guests with errors
- Error type column
- Suggested actions column

4. Statistics Sheet

- Invitation type breakdown
- Time-based analysis
- Pivot tables

5. Timeline Sheet

- Chronological events
- Milestone tracking

6. Messages Sheet

- Reminder log
- Thank you log
- Custom messages sent

CSV Export (Simple Format)

Single file with all data:

```
ID,Name,WhatsApp>Type,Status,Sent_Date,Delivered_Date>Error_Type>Error_Message
1,Raj Patel,+919998118489,VIP,delivered,2024-12-10 14:30,2024-12-10 14:31
3,Amit Kumar,+911234567890,Standard,error,2024-12-10 14:32,,invalid_number
...
```

Real-Time Updates

Status Synchronization:

- Guest table updates automatically when WhatsApp webhook receives delivery status
- Visual notification when status changes
- Updated count badges in filters
- No page refresh required

WebSocket/Polling (implementation detail):

- Poll every 5 seconds during active sending
- WebSocket for instant updates (preferred)

- Visual indicator: "Updating..." badge
-

Performance Considerations

Large Guest Lists (>500 guests):

- Pagination: 50 guests per page
- Virtual scrolling for smooth performance
- Background loading of PDF previews
- Lazy loading of images

Bulk Operations:

- Progress indicators for all bulk actions
 - Cancel option for long-running operations
 - Background processing for large batches
 - Email notification when complete
-

Dashboard & Analytics

Super Admin Dashboard

Key Metrics:

- Total Members (all users)
- Total Customers (all customers)
- Total Admins (admin users)
- Total Events (all events)
- Cards Sent (system-wide)
- Cards Delivered (system-wide)
- Delivery Rate (percentage)
- Active Events (in-progress)
- Completed Events
- Pending Approvals (demos awaiting approval)
- Failed Deliveries

Charts:

- Events over time (line chart)
- Cards sent by admin (bar chart)
- Delivery success rate (pie chart)

- Event types distribution (donut chart)
- Monthly revenue/activity (line chart)

Recent Activity Feed:

- New customer assignments
 - Events created
 - Demos approved/rejected
 - Bulk sends completed
 - Delivery failures
-

Admin Dashboard

Key Metrics (filtered to Admin's customers):

- My Customers
- My Events
- Cards Sent (by me)
- Cards Delivered
- Delivery Rate
- Active Events (mine)
- Completed Events (mine)
- Pending Approvals (my demos)

Charts:

- My events over time
- My delivery success rate
- My customers by event type

Quick Actions:

- Create new event
 - View pending approvals
 - Check failed deliveries
 - View recent notifications
-

WhatsApp Integration System

WhatsApp Cloud API Setup

Prerequisites:

1. Facebook Business Account
2. WhatsApp Business Account
3. Phone number registered with WhatsApp
4. Access token from Meta for Developers

Configuration:

```
WHATSSAPP_API_URL=https://graph.facebook.com/v18.0/123456789/messages  
WHATSSAPP_API_TOKEN=your_access_token_here  
WHATSSAPP_PHONE_NUMBER_ID=your_phone_number_id
```

Setup Steps:

1. Create Facebook Business Account
2. Add WhatsApp product to Business Account
3. Register phone number
4. Get permanent access token
5. Configure webhook for delivery status
6. Add token and phone number ID to settings

Message Templates

Default Template:

Hello {name} !

You're cordially invited to our {event_type}.

 Date: {date}
 Venue: {venue}

Please find your personalized invitation card attached.

Looking forward to celebrating with you!

Custom Templates (configurable in settings):

- Wedding Template
- Birthday Template
- Corporate Template
- Anniversary Template

Template Variables:

- `{name}` : Guest name from CSV
 - `{event_type}` : Event type
 - `{date}` : Event date
 - `{venue}` : Event venue
 - `{custom_message}` : Admin's custom message
 - Any CSV column can be used as variable
-

Delivery Tracking

Status Flow:

pending → sent → delivered → read
↓
failed

Status Definitions:

- **pending**: Queued for sending
- **sent**: Message sent to WhatsApp server
- **delivered**: Message delivered to recipient's device
- **read**: Recipient opened the message
- **failed**: Delivery failed (invalid number, blocked, etc.)

Webhook Events:

WhatsApp sends webhook callbacks for status updates:

```
{
  "entry": [
    {
      "changes": [
        {
          "value": {
            "statuses": [
              {
                "id": "wamid.HBgNOTE5OTk4...",
                "status": "delivered",
                "timestamp": "1702396800",
                "recipient_id": "919998118489"
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Error Handling

Common Errors:

Error Code	Description	Solution
131026	Message undeliverable	Check phone number format
131047	Re-engagement message	User hasn't messaged business in 24h
131053	Media upload failed	Check file size/format
133000	Rate limit hit	Reduce send rate
133005	Access denied	Check access token

Retry Logic:

- Failed sends are marked with error status
- Admin can manually retry from Customer Tab
- Automatic retry for transient errors (network issues)
- Permanent failures (invalid number) are not retried

PDF Management & Delivery

PDF Lifecycle

PDF LIFECYCLE

1. TEMPLATE UPLOAD
↳ Stored in: /storage/uploads/{customer_id}/{event_id}/template.pdf
2. PREVIEW GENERATION
↳ Images in: /storage/previews/{customer_id}/{event_id}/page_*.png
3. DEMO GENERATION
↳ PDF in: /storage/outputs/{customer_id}/{event_id}/demo.pdf
4. BATCH GENERATION
↳ PDFs in:

/storage/outputs/{customer_id}/{event_id}/{guest_id}_card.pdf

5. WHATSAPP UPLOAD

↳ PDF uploaded to WhatsApp Media API (temporary)

6. DELIVERY

↳ WhatsApp sends PDF to recipient

7. POST-EVENT

↳ Option 1: Keep PDFs (for records)

↳ Option 2: Delete PDFs (auto-cleanup)

Storage Structure

```
storage/
└── uploads/
    └── {customer_id}/
        └── {event_id}/
            ├── template.pdf
            └── guests.csv

    └── previews/
        └── {customer_id}/
            └── {event_id}/
                ├── page_1.png
                ├── page_2.png
                └── demo_preview.png

    └── outputs/
        └── {customer_id}/
            └── {event_id}/
                ├── demo.pdf
                ├── 1_card.pdf
                ├── 2_card.pdf
                ├── 3_card.pdf
                └── ...
```

Duplicate Prevention

Rule: Never send the same PDF twice to the same guest.

Implementation:

```
def can_send_to_guest(guest_id: int) -> bool:
    """Check if guest is eligible to receive card"""
    with get_cursor() as cursor:
        cursor.execute("""
            SELECT sent_status FROM guests WHERE id = ?
        """, (guest_id,))
        status = cursor.fetchone()[0]

    # Can only send if pending, failed, or error
    return status in ('pending', 'failed', 'error')
```

UI Indication:

```
// Frontend shows send button only for eligible guests
{guest.sent_status === 'pending' ||
guest.sent_status === 'failed' ||
guest.sent_status === 'error' ? (
    <button onClick={() => sendCard(guest.id)}>Send</button>
) : (
    <span className="text-green-500">✓ Sent</span>
)}
```

Auto-Cleanup

Cleanup Trigger: When event is marked as "completed"

Cleanup Options:

```
{
  "delete_pdfs": true,           // Delete generated PDFs
  "delete_template": false,      // Keep template for reference
  "delete_csv": false,           // Keep guest list
  "archive_data": true,          // Move to archive folder
  "cleanup_after_days": 30       // Auto-cleanup after 30 days
}
```

Cron Job (optional):

```

# Run daily at 2 AM to cleanup old events
0 2 * * * python scripts/cleanup_old_files.py

# scripts/cleanup_old_files.py
import os
from datetime import datetime, timedelta
from backend.database.database import get_cursor

def cleanup_old_events():
    """Delete PDFs for events completed more than 30 days ago"""
    cutoff_date = datetime.now() - timedelta(days=30)

    with get_cursor() as cursor:
        cursor.execute("""
            SELECT id, customer_id FROM events
            WHERE status = 'completed'
            AND completed_at < ?
            AND pdfs_deleted = 0
        """, (cutoff_date,))

        events = cursor.fetchall()

        for event in events:
            event_dir = f"storage/outputs/{event['customer_id']}/{event['id']}"
            if os.path.exists(event_dir):
                # Delete all PDFs
                for file in os.listdir(event_dir):
                    if file.endswith('.pdf'):
                        os.remove(os.path.join(event_dir, file))

                # Mark as deleted
                cursor.execute("""
                    UPDATE events SET pdfs_deleted = 1 WHERE id = ?
                """, (event['id'],))

            print(f"Cleaned up event {event['id']}")


```

Notification System

Notification Types

Type	Trigger	Recipient	Priority
customer_assigned	Super Admin assigns customer	Admin	Medium
inquiry_received	New inquiry submitted	Super Admin	High
demo_ready	Admin creates demo	Customer	High
demo_approved	Customer approves demo	Admin	High
demo_rejected	Customer rejects demo	Admin	High
generation_complete	Batch generation done	Admin	Medium
send_complete	WhatsApp batch sent	Admin	Medium
send_failed	Multiple send failures	Admin	High
event_completed	Event marked complete	Admin, Customer	Low
password_reset	Password reset requested	User	High
account_approved	Admin account approved	Admin	High

Notification Delivery Channels

In-App Notifications:

- Displayed in UI notification bell icon
- Stored in `notifications` table
- Real-time updates via polling or WebSocket

Email Notifications (configurable):

- High-priority notifications sent via email
- Uses SMTP configuration from settings
- Template-based emails

WhatsApp Notifications (future):

- Critical alerts via WhatsApp
- Requires separate WhatsApp Business API setup

Notification Management

Batch Mark as Read:

```
UPDATE notifications
SET is_read = 1
```

```
WHERE user_id = ?  
AND is_read = 0;
```

Auto-Cleanup:

```
-- Delete read notifications older than 90 days  
DELETE FROM notifications  
WHERE is_read = 1  
AND created_at < datetime('now', '-90 days');
```

Settings & Configuration

Global Settings (Super Admin)

WhatsApp Configuration:

- API URL
- Access Token
- Phone Number ID
- Message templates
- Rate limiting

Email Configuration:

- SMTP host
- SMTP port
- SMTP username/password
- From email address
- Email templates

File Storage:

- Max upload size
- Auto-delete after days
- Storage quota per customer

System Settings:

- Timezone
- Date format
- Currency (for billing features)
- Default language

User Settings (All Users)

Personal Preferences:

- Theme (light/dark)
- Notifications enabled
- Email notifications
- Dashboard view
- Language preference

Admin-Specific Settings:

- Default event type
- Auto-assign customers
- Email signature
- Working hours

Workflows & Business Logic

Demo Approval Workflow

DEMO APPROVAL WORKFLOW

1. Admin creates demo
 - └> System generates demo PDF
 - └> Notification sent to Customer (WhatsApp/Email)
 - └> Demo status: "pending_approval"
2. Customer reviews demo
 - ├> APPROVE
 - └> Notification to Admin
 - └> Proceed to batch generation
 - └> REJECT
 - └> Customer provides feedback
 - └> Notification to Admin
 - └> Admin makes revisions
 - └> Generate new demo
 - └> Loop back to step 2

Database Updates:

```
-- Approve demo
UPDATE events
SET demo_approved = 1,
    status = 'active'
WHERE id = ?;

-- Reject demo
UPDATE events
SET demo_approved = 0,
    demo_rejection_reason = ?
WHERE id = ?;
```

Zone Conflict Resolution Workflow

1. Admin defines zones
 - ↳ System detects overlaps
 - ↳ No conflicts: Save zones
 - |
 - ↳ Conflicts detected:
 - ↳ Show warning modal
 - ↳ User adjusts zones
 - |
 - ↳ Re-check conflicts
 - |
 - ↳ User continues anyway
 - ↳ Save with warning flag

Conflict Detection Algorithm:

```
for each pair of zones:
    if zones are on same page:
        calculate intersection area
        if intersection > 0:
            calculate overlap percentage
            determine severity (high/medium/low)
            add to conflicts list

return conflicts
```

File Storage System

Storage Quotas (Optional)

```
# config.py
STORAGE_QUOTAS = {
    "sudo_admin": None,          # Unlimited
    "admin": 10 * 1024 * 1024 * 1024,  # 10GB per admin
    "customer": 1 * 1024 * 1024 * 1024  # 1GB per customer
}
```

File Naming Conventions

Template PDFs:

/storage/uploads/{customer_id}/{event_id}/template.pdf

Guest CSV:

/storage/uploads/{customer_id}/{event_id}/guests.csv

Demo PDF:

/storage/outputs/{customer_id}/{event_id}/demo.pdf

Generated PDFs:

/storage/outputs/{customer_id}/{event_id}/{guest_id}_card.pdf

Preview Images:

/storage/previews/{customer_id}/{event_id}/page_{page_number}.png

Backup Strategy

Backup Scope:

- Database: Daily backup
- Templates & CSVs: Weekly backup
- Generated PDFs: Optional (can be regenerated)

Backup Script:

```
#!/bin/bash
# scripts/backup_database.sh

BACKUP_DIR="backups"
DATE=$(date +%Y%m%d_%H%M%S)

# Backup database
sqlite3 invitewala.db ".backup $BACKUP_DIR/invitewala_$DATE.db"

# Backup critical files
tar -czf $BACKUP_DIR/templates_$DATE.tar.gz storage/uploads/

# Keep last 30 days of backups
find $BACKUP_DIR -name "*.db" -mtime +30 -delete
find $BACKUP_DIR -name "*.tar.gz" -mtime +30 -delete
```

Deployment Guide

Development Environment

Prerequisites:

- Python 3.10+
- Node.js 18+
- SQLite 3.35+

Setup Steps:

1. Clone repository:

```
git clone <repo_url>
cd Project_B
```

2. Backend setup:

```
# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

```
# Initialize database
python -c "from backend.database.database import init_db; init_db()"

# Create .env file
cp .env.example .env
# Edit .env with your configuration
```

3. Frontend setup:

```
cd frontend
npm install
```

4. Start development servers:

Terminal 1(Backend):

```
uvicorn backend.main:app --reload --port 5234
```

Terminal 2 (Frontend - optional, for hot reload):

```
cd frontend
npm run dev
```

Access application:

- Frontend (dev): <http://localhost:5173>
 - Backend API: <http://localhost:5234>
 - API Docs: <http://localhost:5234/api/docs>
-

Production Deployment

Option 1: Traditional Server

1. Build frontend:

```
cd frontend
npm run build
```

2. Configure environment:

```
# Create .env file with production settings
SECRET_KEY=<generate-strong-secret-key>
WHATAPP_API_TOKEN=<your-token>
WHATAPP_PHONE_NUMBER_ID=<your-id>
SMTP_HOST=smtp.gmail.com
SMTP_USERNAME=<your-email>
SMTP_PASSWORD=<your-password>
```

3. Install as systemd service:

```
# /etc/systemd/system/invitewala.service
[Unit]
Description=Invitewala Platform
After=network.target

[Service]
Type=simple
User=www-data
WorkingDirectory=/var/www/invitewala
Environment="PATH=/var/www/invitewala/venv/bin"
ExecStart=/var/www/invitewala/venv/bin/uvicorn backend.main:app --host 0.0.0.0
Restart=always

[Install]
WantedBy=multi-user.target

sudo systemctl enable invitewala
sudo systemctl start invitewala
```

4. Configure Nginx reverse proxy:

```
# /etc/nginx/sites-available/invitewala
server {
    listen 80;
    server_name invitewala.com www.invitewala.com;

    location / {
        proxy_pass http://127.0.0.1:5234;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

```

    proxy_set_header X-Forwarded-Proto $scheme;
}

location /storage {
    alias /var/www/invitewala/storage;
}

client_max_body_size 20M;
}

```

5. Setup SSL with Let's Encrypt:

```

sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d invitewala.com -d www.invitewala.com

```

Option 2: Docker Deployment

Dockerfile:

```

FROM python:3.11-slim

# Install system dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY . .

# Build frontend
RUN apt-get update && apt-get install -y nodejs npm \
    && cd frontend && npm install && npm run build \
    && apt-get remove -y nodejs npm && rm -rf /var/lib/apt/lists/*

# Create storage directories
RUN mkdir -p storage/uploads storage/outputs storage/previews logs

```

```
# Initialize database
RUN python -c "from backend.database.database import init_db; init_db()"

EXPOSE 5234
```

```
CMD ["uvicorn", "backend.main:app", "--host", "0.0.0.0", "--port", "5234"]
```

docker-compose.yml:

```
version: '3.8'
```

services:

invitewala:

```
build: .
```

ports:

```
- "5234:5234"
```

volumes:

```
- ./storage:/app/storage
```

```
- ./logs:/app/logs
```

```
- ./invitewala.db:/app/invitewala.db
```

environment:

```
- SECRET_KEY=${SECRET_KEY}
```

```
- WHATSAPP_API_TOKEN=${WHATSAPP_API_TOKEN}
```

```
- WHATSAPP_PHONE_NUMBER_ID=${WHATSAPP_PHONE_NUMBER_ID}
```

```
- SMTP_HOST=${SMTP_HOST}
```

```
- SMTP_USERNAME=${SMTP_USERNAME}
```

```
- SMTP_PASSWORD=${SMTP_PASSWORD}
```

```
restart: always
```

nginx:

```
image: nginx:alpine
```

ports:

```
- "80:80"
```

```
- "443:443"
```

volumes:

```
- ./nginx.conf:/etc/nginx/nginx.conf
```

```
- ./storage:/usr/share/nginx/html/storage
```

```
- ./certbot/conf:/etc/letsencrypt
```

```
- ./certbot/www:/var/www/certbot
```

depends_on:

```
- invitewala
```

```
restart: always
```

Deploy with Docker:

```
# Build and start  
docker-compose up -d  
  
# View logs  
docker-compose logs -f  
  
# Stop  
docker-compose down
```

Environment Variables

Required Variables:

```
# Application  
SECRET_KEY=your-secret-key-min-32-chars  
  
# WhatsApp  
WHATAPP_API_URL=https://graph.facebook.com/v18.0/{phone_number_id}/messa  
WHATAPP_API_TOKEN=your_whatsapp_access_token  
WHATAPP_PHONE_NUMBER_ID=your_phone_number_id  
  
# Email  
SMTP_HOST=smtp.gmail.com  
SMTP_PORT=587  
SMTP_USERNAME=your_email@gmail.com  
SMTP_PASSWORD=your_app_password  
SMTP_FROM_EMAIL=noreply@invitewala.com  
  
# Database (optional, defaults to SQLite)  
DATABASE_URL=sqlite:///./invitewala.db
```

Troubleshooting

Common Issues & Solutions

1. Login Issues

Issue: "Invalid email or password"

- **Solution:** Verify credentials. Default: admin@invitewala.com / admin123
- **Check:** User exists and `is_verified = 1`

Issue: "Email not verified"

- **Solution:** Manually verify in database:

```
UPDATE users SET is_verified = 1 WHERE email = 'user@example.com';
```

2. PDF Issues

Issue: Zone not appearing on generated PDF

- **Cause:** Coordinate mismatch (pixel vs points)
- **Solution:** Verify conversion `pdf_points = pixel_coords * (72/150)`

Issue: Text overlapping or misaligned

- **Cause:** Zone conflicts or incorrect alignment settings
- **Solution:** Use conflict detector, adjust zone positions

Issue: Blank PDFs generated

- **Cause:** CSV column mapping incorrect
 - **Solution:** Verify zone `csv_column` matches actual CSV column names
-

3. WhatsApp Issues

Issue: "Message undeliverable"

- **Cause:** Invalid phone number format
- **Solution:** Use international format (+919998118489)

Issue: Rate limit exceeded

- **Cause:** Too many messages sent too quickly
- **Solution:** Reduce batch size or implement delays between sends

Issue: Media upload failed

- **Cause:** PDF too large (>16MB for WhatsApp)
 - **Solution:** Compress PDF or split into pages
-

4. Database Issues

Issue: "Database is locked"

- **Cause:** SQLite doesn't handle concurrent writes well
- **Solution:** Implement connection pooling or consider PostgreSQL for production

Issue: "UNIQUE constraint failed"

- **Cause:** Duplicate entry (email, zone ID, etc.)
 - **Solution:** Check for existing records before insert
-

5. File Upload Issues

Issue: "File too large"

- **Cause:** Exceeds MAX_UPLOAD_SIZE
- **Solution:** Increase limit in config or compress file

Issue: "Invalid file type"

- **Cause:** Non-PDF file uploaded as template
 - **Solution:** Validate file extension and MIME type on frontend
-

Debug Mode

Enable verbose logging:

```
uvicorn backend.main:app --reload --log-level debug
```

Check logs:

```
tail -f logs/app.log
tail -f logs/error.log
tail -f logs/whatsapp.log
```

Performance Optimization

Database Optimization:

```
-- Add indexes for frequently queried columns
CREATE INDEX IF NOT EXISTS idx_guests_event_status ON guests(event_id, se
CREATE INDEX IF NOT EXISTS idx_events_customer_status ON events(customer_
```

Caching (optional, with Redis):

```
from functools import lru_cache

@lru_cache(maxsize=100)
def get_customer_events(customer_id: int):
    # Cache customer events
    pass
```

Async Operations (for large batches):

```
from fastapi import BackgroundTasks

@router.post("/generate-batch")
async def generate_batch(event_id: int, background_tasks: BackgroundTasks):
    # Queue generation in background
    background_tasks.add_task(generate_all_pdfs, event_id)
    return {"message": "Generation started", "task_id": "..."}
```

Default Credentials

Email	Password	Role	Notes
admin@invitewala.com	admin123	sudo_admin	Created on first database init

Security Note: Change default password immediately in production!

License

MIT License - © 2024 Invitewala Platform

Support & Contact

- **Email:** support@invitewala.com
 - **Documentation:** <https://docs.invitewala.com>
 - **GitHub Issues:** /issues
 - **Community:** Discord/Slack invite link
-

End of Documentation

Version 2.0 | Last Updated: December 24, 2025