Configuration Guide - Automated Phone Answering System

Table of Contents

Environment Variables Reference

Required Variables

Variable	Description	Example	Notes
`VONAGE_API_KEY`	Your Vonage API key	`a1b2c3d4`	From Vonage Dashboard
`VONAGE_API_SECRET`	Your Vonage API secret	`AbCdEfGhIjKlMnOp`	Keep secure
`VONAGE_APPLICATION_ID`	Voice application UUID	`12345678-1234-1234-1234-123456789012`	Created in dashboard
`VONAGE_PRIVATE_KEY_PATH`	Path to private key file	`./private.key`	Downloaded from Vonage
`VONAGE_PHONE_NUMBER`	Your Vonage phone number	`+15551234567`	Include country code
`GOOGLE_CALENDAR_ID`	Target calendar ID	`primary` or specific ID	From calendar settings
`GOOGLE_CREDENTIALS_PATH`	Service account credentials	`./credentials.json`	From Google Cloud

Essential Configuration

Variable	Description	Default	Options
`STAFF_PHONE_NUMBER`	Escalation contact number	Required	`+15551234567`
`FACILITY_TIMEZONE`	Business timezone	`America/New_York`	Any valid IANA timezone
`BUSINESS_HOURS_START`	Opening hour (24h format)	`9`	0-23
`BUSINESS_HOURS_END`	Closing hour (24h format)	`21`	0-23

Optional Configuration

Variable	Description	Default	Notes
`FLASK_ENV`	Flask environment	`production`	`development` or `production`
`PORT`	Application port	`5000`	1024-65535
`BASE_URL`	Public base URL	`http://localhost:5000`	Used for webhooks
`HOLD_MUSIC_URL`	URL for hold music	None	Optional MP3 URL
`ESCALATION_LOG_FILE`	Escalation log path	`/tmp/escalations.log`	Full file path
`CALLBACK_LOG_FILE`	Callback log path	`/tmp/callbacks.log`	Full file path
`GOOGLE_TOKEN_PATH`	OAuth token cache	`./token.json`	For user auth (optional)

Vonage Voice API Configuration

1. Account Setup

Create Vonage Account

1. Go to developer.nexmo.com (https://developer.nexmo.com)
2. Sign up for new account or log in
3. Complete account verification
4. Add payment method for phone number rental

API Key and Secret

1. Navigate to “Getting Started” in dashboard

2. Copy your API key and secret
3. Store securely - these are your account credentials

2. Voice Application Creation

Application Configuration

```
{
  "name": "Auto Call System",
  "capabilities": {
    "voice": {
      "webhooks": {
        "answer_url": {
          "address": "https://yourdomain.com/webhooks/answer",
          "http_method": "POST"
        },
        "event_url": {
          "address": "https://yourdomain.com/webhooks/events",
          "http_method": "POST"
        }
      }
    }
  }
}
```

Step-by-Step Creation

1. **Go to Applications**
   - Click “Your Applications” in dashboard
   - Click “Create a new application”

2. **Basic Settings**
   - **Name**: `Auto Call System`
   - **Description**: `Automated phone answering for sports facility`

3. **Voice Configuration**
   - Enable “Voice” capability
   - **Answer URL**: `https://yourdomain.com/webhooks/answer`
   - **Event URL**: `https://yourdomain.com/webhooks/events`
   - **HTTP Method**: POST for both

4. **Generate Keys**
   - Click “Generate public/private key pair”
   - Download the private key file
   - Save as `private.key` in your project directory

5. **Save Application**
   - Click “Create Application”
   - Note the Application ID (UUID format)

3. Phone Number Configuration

Purchase Number

1. Go to “Numbers” → “Buy Numbers”
2. Select country (typically your business location)

3. Choose number type:
   - **Voice**: Required
   - **SMS**: Optional
   - **Mobile vs Landline**: Mobile recommended for better delivery

Link Number to Application

1. Go to “Numbers” → “Your Numbers”
2. Click settings icon next to purchased number
3. Select your Voice application
4. Save configuration

Number Testing

```
# Test number ownership
curl -X GET "https://api.nexmo.com/account/numbers" \
  -u "$VONAGE_API_KEY:$VONAGE_API_SECRET"
```

4. Webhook Security (Recommended)

JWT Verification Setup

Add to your environment:

```
VONAGE_SIGNATURE_SECRET=your_signature_secret
VONAGE_SIGNATURE_METHOD=sha256
```

Implementation in Flask

```
import jwt
from functools import wraps

def verify_vonage_signature(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not request.headers.get('Authorization'):
            return jsonify({'error': 'Missing signature'}), 401

        try:
            token = request.headers['Authorization'].replace('Bearer ', '')
            payload = jwt.decode(
                token,
                os.getenv('VONAGE_SIGNATURE_SECRET'),
                algorithms=['HS256']
            )
            return f(*args, **kwargs)
        except jwt.InvalidTokenError:
            return jsonify({'error': 'Invalid signature'}), 401

    return decorated_function
```

Google Calendar Integration Setup

1. Google Cloud Project Setup

Create Project

1. Go to Google Cloud Console (https://console.cloud.google.com)
2. Click project dropdown → "New Project"
3. **Project Name**: `auto-call-system`
4. **Organization**: Select appropriate org
5. Click "Create"

Enable Calendar API

```
# Using gcloud CLI (optional)
gcloud services enable calendar-json.googleapis.com
```

Or via console:
1. Go to "APIs & Services" → "Library"
2. Search "Google Calendar API"
3. Click "Enable"

2. Service Account Creation

Create Service Account

1. Navigate to "APIs & Services" → "Credentials"
2. Click "Create Credentials" → "Service Account"
3. **Service Account Details**:
   - **Name**: `auto-call-calendar-service`
   - **ID**: `auto-call-calendar-service` (auto-generated)
   - **Description**: `Service account for automated call system calendar integration`

Assign Roles

Standard roles:
- `Calendar API Service Agent` (if available)
- `Editor` (for broader access)

Custom role (recommended):

```
{
  "title": "Calendar Manager",
  "description": "Manages calendar events for auto call system",
  "stage": "GA",
  "includedPermissions": [
    "calendar.calendars.get",
    "calendar.events.list",
    "calendar.events.create",
    "calendar.events.update",
    "calendar.events.delete",
    "calendar.freebusy.query"
  ]
}
```

Generate Key

1. Click on created service account
2. Go to “Keys” tab
3. Click “Add Key” → “Create new key”
4. Choose **JSON** format
5. Download and rename to `credentials.json`

3. Calendar Sharing Configuration

Option A: Primary Calendar Access

For personal/single-user setup:
1. Share your primary calendar with service account email
2. Use `GOOGLE_CALENDAR_ID=primary`

Option B: Dedicated Calendar (Recommended)

1. **Create New Calendar**:
   - Open Google Calendar
   - Click “+” next to “Other calendars”
   - Select “Create new calendar”
   - **Name**: `Sports Facility Bookings`
   - **Description**: `Automated bookings from phone system`

2. **Get Calendar ID**:
   - Click calendar settings (3 dots menu)
   - Scroll to “Integrate calendar”
   - Copy “Calendar ID” (format: `abc123@group.calendar.google.com`)

3. **Share with Service Account**:
   - Go to “Share with specific people”
   - Add service account email (from credentials.json)
   - Permission: “Make changes and manage sharing”

Environment Configuration

```
# For primary calendar
GOOGLE_CALENDAR_ID=primary

# For dedicated calendar
GOOGLE_CALENDAR_ID=abc123@group.calendar.google.com
```

4. Authentication Testing

Test Service Account Access

```python
#!/usr/bin/env python3
"""Test Google Calendar integration"""

import os
from google.oauth2 import service_account
from googleapiclient.discovery import build

def test_calendar_access():
    # Load credentials
    credentials = service_account.Credentials.from_service_account_file(
        'credentials.json',
        scopes=['https://www.googleapis.com/auth/calendar']
    )

    # Build service
    service = build('calendar', 'v3', credentials=credentials)

    # Test calendar access
    try:
        calendar = service.calendars().get(calendarId='primary').execute()
        print(f"✅ Calendar access successful: {calendar['summary']}")

        # Test event listing
        events = service.events().list(calendarId='primary', maxResults=1).execute()
        print(f"✅ Event listing successful: {len(events.get('items', []))} events
found")

        return True
    except Exception as e:
        print(f"❌ Calendar access failed: {e}")
        return False

if __name__ == "__main__":
    test_calendar_access()
```

Run test:

```
cd /path/to/your/project
python3 test_calendar.py
```

5. Calendar Configuration Options

Business Hours Enforcement

```python
# In calendar_helper.py, customize business hours
BUSINESS_HOURS = {
    'start': 9,      # 9 AM
    'end': 21,       # 9 PM
    'timezone': 'America/New_York',
    'days': [0, 1, 2, 3, 4, 5, 6],  # 0=Monday, 6=Sunday
    'holidays': [
        '2025-01-01',  # New Year's Day
        '2025-07-04',  # Independence Day
        '2025-12-25'   # Christmas Day
    ]
}
```

Event Templates

```
# Default event template
DEFAULT_EVENT_TEMPLATE = {
    'summary': 'Basketball Court Rental - {duration}h',
    'description': '''
Booking Details:
- Duration: {duration} hour(s)
- Party Size: {party_size} people
- Contact: {phone_number}
- Booking Method: Automated Phone System
- Rate: ${rate}/hour

Facility Rules:
- Check in 15 minutes early
- Maximum {max_capacity} people
- No outside food or drinks
- Clean up after use
    ''',
    'location': 'Sports Facility - Basketball Court',
    'colorId': '9',  # Blue color
    'reminders': {
        'useDefault': False,
        'overrides': [
            {'method': 'email', 'minutes': 24 * 60},  # 1 day before
            {'method': 'popup', 'minutes': 60}        # 1 hour before
        ]
    }
}
```

Business Logic Configuration

1. Pricing Configuration

Pricing Data Structure

The system uses CSV files in `/pricing_data/` directory:

```
pricing_data/
├── hourly_rates.csv
├── birthday_packages.csv
├── seasonal_adjustments.csv
└── special_rates.csv
```

hourly_rates.csv Format

```
service_type,time_period,base_rate,peak_multiplier,description
basketball,weekday_morning,25.00,1.0,"Standard weekday morning rate"
basketball,weekday_evening,35.00,1.2,"Peak weekday evening rate"
basketball,weekend,40.00,1.3,"Weekend premium rate"
```

Configuration Parameters

```
# In pricing.py
PRICING_CONFIG = {
    'peak_hours': {
        'weekday': {'start': 17, 'end': 21},    # 5 PM - 9 PM
        'weekend': {'start': 9, 'end': 18}      # 9 AM - 6 PM
    },
    'seasonal_multipliers': {
        'summer': 1.2,     # June, July, August
        'winter': 0.9,     # December, January, February
        'standard': 1.0    # Spring, Fall
    },
    'group_discounts': {
        'threshold': 4,    # 4+ hour bookings
        'discount': 0.1    # 10% discount
    }
}
```

2. Business Hours Configuration

Time Zone Handling

```
import pytz
from datetime import datetime, time

# Business configuration
FACILITY_CONFIG = {
    'timezone': pytz.timezone('America/New_York'),
    'business_hours': {
        'monday': {'open': time(9, 0), 'close': time(21, 0)},
        'tuesday': {'open': time(9, 0), 'close': time(21, 0)},
        'wednesday': {'open': time(9, 0), 'close': time(21, 0)},
        'thursday': {'open': time(9, 0), 'close': time(21, 0)},
        'friday': {'open': time(9, 0), 'close': time(21, 0)},
        'saturday': {'open': time(8, 0), 'close': time(22, 0)},
        'sunday': {'open': time(10, 0), 'close': time(20, 0)}
    },
    'holiday_hours': {
        'new_years': {'open': time(12, 0), 'close': time(18, 0)},
        'christmas_eve': {'open': time(9, 0), 'close': time(15, 0)},
        'closed_holidays': ['christmas', 'thanksgiving']
    }
}
```

3. Response Templates Configuration

Greeting Messages

```
# In app.py or separate config file
RESPONSE_TEMPLATES = {
    'greeting': [
        "Hello! Thank you for calling City Sports Center. I'm here to help you with court rentals, pricing, and availability. How can I assist you today?",
        "Hi there! Welcome to City Sports Center. I can help you check pricing, availability, or make a booking. What would you like to know?",
    ],
    'after_hours': [
        "Thank you for calling City Sports Center. We're currently closed. Our business hours are 9 AM to 9 PM, Monday through Sunday. Please call back during business hours or leave a message after the tone.",
    ],
    'pricing_response': [
        "Our basketball court rental is ${rate} per hour {time_description}. {additional_info} Would you like to check availability for a specific time?",
    ],
    'availability_response': [
        "I can check availability for you. {availability_info} Would you like me to hold this time slot or get pricing information?",
    ],
    'booking_confirmation': [
        "Perfect! I have you booked for {duration} hour{s} on {date} from {start_time} to {end_time}. Your total is ${total_cost}. {confirmation_details}",
    ],
    'escalation_transfer': [

"I'll connect you with one of our staff members who can better assist you with that. Please hold while I transfer your call.",
    ]
}
```

Advanced Configuration Options

1. NLU Customization

Intent Recognition Patterns

```
# In nlu.py
CUSTOM_INTENT_PATTERNS = {
    'pricing': [
        r'\b(price|cost|rate|fee|charge|how much|pricing|expensive|cheap)\b',
        r'\b(hourly|per hour|birthday party|package|membership)\b',
        r'\b(what does it cost|how much does|price for)\b'
    ],
    'availability': [
        r'\b(available|availability|free|open|check|vacant)\b',
        r'\b(tomorrow|today|this week|next week|weekend|weekday)\b',
        r'\b(morning|afternoon|evening|night|time slot)\b',
        r'\b(when can|what times|is.*available)\b'
    ],
    'booking': [
        r'\b(book|reserve|schedule|make.*appointment|rent)\b',
        r'\b(want to|would like to|need to|looking to)\b.*\b(book|rent)\b',
        r'\b(for.*hours?|from.*to|at.*time)\b'
    ],
    # Add custom intents here
    'cancel_booking': [
        r'\b(cancel|cancellation|cancel.*booking|cancel.*reservation)\b',
        r'\b(need to cancel|want to cancel|have to cancel)\b'
    ]
}
```

Entity Extraction Patterns

```
ENTITY_PATTERNS = {
    'time_period': {
        'hourly': r'\b(hour|hourly|per hour|by the hour)\b',
        'daily': r'\b(day|daily|all day|full day)\b',
        'package': r'\b(package|birthday|party|event)\b'
    },
    'date_time': {
        'today': r'\b(today|this morning|this afternoon|tonight)\b',
        'tomorrow': r'\b(tomorrow|tomorrow morning|tomorrow evening)\b',
        'this_week': r'\b(this week|later this week)\b',
        'next_week': r'\b(next week|following week)\b',
        'weekend': r'\b(weekend|saturday|sunday|this weekend|next weekend)\b'
    }
}
```

2. Voice Response Customization

TTS Voice Configuration

```
# Vonage TTS options
TTS_CONFIG = {
    'language': 'en-US',
    'style': 'neural',      # More natural sounding
    'voice_name': 'Amy',    # Female voice
    'speed': 0,             # Normal speed (-10 to 10)
    'volume': 0,            # Normal volume (-10 to 10)
    'text_type': 'text'     # or 'ssml' for advanced control
}

# SSML example for more control
SSML_GREETING = """
<speak>
    <prosody rate="medium" pitch="medium">
        Hello! <break time="0.5s"/>
        Thank you for calling City Sports Center.
        <emphasis level="moderate">I'm here to help you</emphasis>
        with court rentals, pricing, and availability.
        <break time="0.3s"/>
        How can I assist you today?
    </prosody>
</speak>
"""
```

3. Logging and Monitoring Configuration

Structured Logging Setup

```
import logging
import json
from datetime import datetime

class JSONFormatter(logging.Formatter):
    def format(self, record):
        log_entry = {
            'timestamp': datetime.utcnow().isoformat(),
            'level': record.levelname,
            'message': record.getMessage(),
            'module': record.module,
            'function': record.funcName,
            'line': record.lineno
        }

        # Add call-specific context if available
        if hasattr(record, 'call_uuid'):
            log_entry['call_uuid'] = record.call_uuid
        if hasattr(record, 'caller_number'):
            log_entry['caller_number'] = record.caller_number

        return json.dumps(log_entry)

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    handlers=[
        logging.FileHandler('/var/log/auto-call-system/app.log'),
        logging.StreamHandler()
    ]
)
```

Metrics Collection

```
# In app.py
METRICS = {
    'calls_answered': 0,
    'escalations': 0,
    'bookings_made': 0,
    'average_call_duration': 0,
    'intent_accuracy': {}
}

def update_metrics(metric_name, value=1):
    """Update system metrics"""
    METRICS[metric_name] += value

    # Log metrics periodically or on request
    if METRICS['calls_answered'] % 10 == 0:
        logging.info(f"Metrics update: {METRICS}")
```

Configuration Validation

1. Environment Validation Script

Create `validate_config.py`:

```
#!/usr/bin/env python3
"""Configuration validation script"""

import os
import sys
from pathlib import Path
import json

def validate_environment():
    """Validate all required environment variables"""
    required_vars = [
        'VONAGE_API_KEY',
        'VONAGE_API_SECRET',
        'VONAGE_APPLICATION_ID',
        'VONAGE_PRIVATE_KEY_PATH',
        'VONAGE_PHONE_NUMBER',
        'GOOGLE_CALENDAR_ID',
        'GOOGLE_CREDENTIALS_PATH',
        'STAFF_PHONE_NUMBER'
    ]

    missing_vars = []
    for var in required_vars:
        if not os.getenv(var):
            missing_vars.append(var)

    if missing_vars:
        print(f"❌ Missing required environment variables: {', '.join(missing_vars)}")
        return False

    print("✅ All required environment variables are set")
    return True

def validate_files():
    """Validate required files exist"""
    required_files = [
        os.getenv('VONAGE_PRIVATE_KEY_PATH', './private.key'),
        os.getenv('GOOGLE_CREDENTIALS_PATH', './credentials.json')
    ]

    missing_files = []
    for file_path in required_files:
        if not Path(file_path).exists():
            missing_files.append(file_path)

    if missing_files:
        print(f"❌ Missing required files: {', '.join(missing_files)}")
        return False

    print("✅ All required files are present")
    return True

def validate_google_credentials():
    """Validate Google credentials format"""
    creds_path = os.getenv('GOOGLE_CREDENTIALS_PATH', './credentials.json')

    try:
        with open(creds_path, 'r') as f:
            creds = json.load(f)

        required_fields = ['type', 'project_id', 'private_key', 'client_email']
        missing_fields = [field for field in required_fields if field not in creds]
```

```
        if missing_fields:
            print(f"❌ Google credentials missing fields: {',
'.join(missing_fields)}")
            return False

        if creds['type'] != 'service_account':
            print("❌ Google credentials must be service account type")
            return False

        print("✅ Google credentials format is valid")
        return True

    except Exception as e:
        print(f"❌ Error validating Google credentials: {e}")
        return False

def validate_business_config():
    """Validate business configuration"""
    try:
        start_hour = int(os.getenv('BUSINESS_HOURS_START', 9))
        end_hour = int(os.getenv('BUSINESS_HOURS_END', 21))

        if not 0 <= start_hour <= 23:
            print("❌ BUSINESS_HOURS_START must be 0-23")
            return False

        if not 0 <= end_hour <= 23:
            print("❌ BUSINESS_HOURS_END must be 0-23")
            return False

        if start_hour >= end_hour:
            print("❌ BUSINESS_HOURS_START must be less than BUSINESS_HOURS_END")
            return False

        print("✅ Business hours configuration is valid")
        return True

    except ValueError:
        print("❌ Business hours must be integers")
        return False

if __name__ == "__main__":
    print("Validating Auto Call System Configuration...")
    print("-" * 50)

    checks = [
        validate_environment(),
        validate_files(),
        validate_google_credentials(),
        validate_business_config()
    ]

    if all(checks):
        print("\n🎉 All configuration checks passed!")
        sys.exit(0)
    else:
        print("\n❌ Configuration validation failed!")
        sys.exit(1)
```

2. Run Validation

```
# Make script executable
chmod +x validate_config.py

# Run validation
python3 validate_config.py
```

3. Integration Test

Create `test_integration.py`:

```
#!/usr/bin/env python3
"""Integration testing for all components"""

from calendar_helper import CalendarHelper
from pricing import PricingEngine
from nlu import SportsRentalNLU
from escalation import EscalationHandler

def test_all_components():
    """Test all system components"""

    print("Testing Calendar Integration...")
    try:
        calendar = CalendarHelper()
        calendar.check_availability()
        print("✅ Calendar integration working")
    except Exception as e:
        print(f"❌ Calendar integration failed: {e}")

    print("\nTesting Pricing Engine...")
    try:
        pricing = PricingEngine()
        rate = pricing.get_pricing_info('basketball', 'hourly')
        print(f"✅ Pricing engine working: {rate}")
    except Exception as e:
        print(f"❌ Pricing engine failed: {e}")

    print("\nTesting NLU...")
    try:
        nlu = SportsRentalNLU()
        result = nlu.process_speech_input("How much does it cost?", {})
        print(f"✅ NLU working: Intent = {result['intent']}")
    except Exception as e:
        print(f"❌ NLU failed: {e}")

    print("\nTesting Escalation Handler...")
    try:
        escalation = EscalationHandler()
        ncco = escalation.create_escalation_ncco('payment_issue', {})
        print("✅ Escalation handler working")
    except Exception as e:
        print(f"❌ Escalation handler failed: {e}")

if __name__ == "__main__":
    test_all_components()
```

Configuration Complete! 🎉

Your system is now properly configured. Proceed to the customization guide to tailor responses and pricing to your specific needs.