ACE Platform - External Services Integration Guide

Document Version: 1.0 **Last Updated:** December 7, 2025 **Platform:** ACE - Academic Career Enhancement

---

Table of Contents

Overview

The ACE platform integrates with 9 external services to provide a comprehensive educational and career development experience. This document details the configuration, flow, and implementation of each service.

Architecture Pattern: - Backend: Laravel PHP (API) - Frontend: Next.js (React) - Communication: RESTful APIs, Webhooks, WebSockets

---

1. DiDit Service - Identity Verification

Purpose

Verifies university student and teacher IDs during the registration process to ensure authenticity and prevent fraud.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 38-43)

Environment Variables:

```
DIDIT_API_KEY=your_api_key_here
DIDIT_WEBHOOK_SECRET=your_webhook_secret
DIDIT_WORKFLOW_ID=your_workflow_id
```

Frontend Environment:

```
NEXT_PUBLIC_DIDIT_API_KEY=your_public_api_key
```

Implementation Flow

```
┌─────────────┐
│    User     │
│   Signup    │
└──────┬──────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 1. Frontend calls                           │
│    POST /api/didit/create-session           │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. DiDit session created                    │
│    - Workflow ID assigned                   │
│    - Callback URL configured                │
│    - Session ID returned                    │
└──────┬──────────────────────────────────────┘
       │
       ▼
```

```
┌─────────────────────────────────────────────┐
│ 3. User redirected to DiDit platform        │
│    - Uploads ID documents                   │
│    - Completes verification steps           │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 4. DiDit processes verification             │
│    - Document validation                    │
│    - Identity verification                  │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 5. DiDit sends webhook                      │
│    POST /api/didit/webhook                  │
│    - Signature verification (HMAC-SHA256)   │
│    - Verification results                   │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 6. Frontend polls status                    │
│    GET /api/didit/session-status/{id}       │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 7. Verification data stored                 │
│    - User profile updated                   │
│    - Registration proceeds                  │
└─────────────────────────────────────────────┘
```

Key Integration Files

- `frontend/pages/api/didit/create-session.ts` - Creates verification sessions
- `frontend/pages/api/didit/session-status/[sessionId].ts` - Status checking
- `frontend/pages/api/didit/webhook.ts` - Webhook handler
- `frontend/app/signup/page.tsx` - Registration UI integration

Authentication

API Authentication: - Header: `X-Api-Key: {DIDIT_API_KEY}`

Webhook Security: - HMAC-SHA256 signature verification - Secret key: `DIDIT_WEBHOOK_SECRET`

Error Handling

- **401 Unauthorized:** Invalid API key
- **404 Not Found:** Session not found
- **400 Bad Request:** Invalid webhook signature
- Comprehensive logging for debugging

Cost Considerations

- **Type:** Paid service
- **Billing:** Per verification request
- **Recommendation:** Monitor usage and set alerts

---

2. MailGun Service - Email Delivery

Purpose

Transactional email delivery system for all platform notifications, ensuring reliable communication with users.

Configuration

Backend Configuration File: `backend/config/mail.php` (lines 6-11, 47-52)

Environment Variables:

```
MAIL_MAILER=mailgun
MAILGUN_DOMAIN=your-domain.mailgun.org
MAILGUN_SECRET=your_mailgun_api_key
MAILGUN_ENDPOINT=api.mailgun.net
```

Composer Dependency:

```
"symfony/mailgun-mailer": "^6.0"
```

Email Types & Templates

1. Welcome Notification

File: `backend/app/Notifications/WelcomeNotification.php` - Sent: Upon successful registration - Languages: Bilingual (Arabic + English) - Content: Welcome message, getting started guide

2. Teacher Approval Notification

File: `backend/app/Notifications/TeacherApprovedNotification.php` - Sent: When admin approves teacher account - Content: Approval confirmation, next steps

3. Teacher Rejection Notification

File: `backend/app/Notifications/TeacherRejectedNotification.php` - Sent: When admin rejects teacher application - Content: Rejection notice, reason (if provided)

4. Job Application Notification

File: `backend/app/Notifications/NewJobApplication.php` - Sent to: Company HR - Trigger: Student applies for job - Content: Applicant details, resume link

5. Application Status Update

File: `backend/app/Notifications/ApplicationStatusUpdated.php` - Sent to: Student - Trigger: Application status changes - Content: New status, company details

6. New Job Posted

File: `backend/app/Notifications/NewJobPosted.php` - Sent to: Relevant students - Trigger: New job matching criteria - Content: Job details, apply link

Implementation Flow

```
┌─────────────────────────────────────────────┐
│ 1. Event Triggered                          │
│    (Registration, Approval, Application)    │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Laravel Notification Dispatched          │
│    Notification::send($user, $notification) │
```

```
└──────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────┐
│ 3. Notification Class Builds Email           │
│    - toMail() method                         │
│    - Subject (AR + EN)                       │
│    - Body content                            │
│    - Action buttons                          │
└──────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────┐
│ 4. Laravel Mail Facade                       │
│    Mail::to($recipient)->send($mailable)     │
└──────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────┐
│ 5. MailGun Transport Layer                   │
│    - API call to MailGun                     │
│    - Domain: MAILGUN_DOMAIN                  │
│    - Auth: MAILGUN_SECRET                    │
└──────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────┐
│ 6. MailGun Processes & Delivers              │
│    - Queuing                                 │
│    - Delivery attempt                        │
│    - Bounce/complaint handling               │
└──────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────┐
│ 7. Delivery Confirmation                     │
│    - Logged in Laravel logs                  │
│    - Visible in MailGun dashboard            │
└──────────────────────────────────────────────┘
```

Best Practices

1. **Bilingual Support:** All emails include both Arabic and English
2. **Clear CTAs:** Action buttons for important next steps
3. **Logging:** All email attempts logged for troubleshooting
4. **Queue Management:** Use Laravel queues for async sending
5. **Error Handling:** Graceful degradation on failures

Cost Considerations

- **Free Tier:** 5,000 emails/month for 3 months
- **Flex Plan:** Pay-as-you-go ($0.80/1000 emails)
- **Recommendation:** Monitor monthly volume

---

3. Gemini Service - AI Career Guidance

Purpose

Google's Gemini AI integration for intelligent career counseling, CV analysis, and personalized learning path recommendations.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 78-82)

Environment Variables:

```
GEMINI_API_KEY=your_google_api_key
GEMINI_MODEL=gemini-1.5-flash
GEMINI_BASE_URL=https://generativelanguage.googleapis.com/v1beta/models
```

API Endpoints & Features

1. General Career Chat

Endpoint: `POST /api/ai-career/chat`

Request:

```
{
  "message": "What career paths are suitable for my profile?"
}
```

Response:

```
{
  "success": true,
  "data": {
    "response": "Based on your Computer Science background...",
    "conversation_id": 123
  }
}
```

2. CV Analysis

Endpoint: `POST /api/ai-career/analyze-cv`

Request:

```
{
  "cv_text": "Full CV content or URL"
}
```

Features: - Strengths identification - Weaknesses analysis - Improvement suggestions - ATS optimization tips

3. Personalized Learning Path

Endpoint: `POST /api/ai-career/learning-path`

Request:

```
{
  "target_role": "Full Stack Developer"
}
```

Features: - Skills gap identification - Course recommendations - Learning timeline - Resource suggestions

4. Job Recommendations

Endpoint: `POST /api/ai-career/job-recommendations`

Features: - Role matching based on profile - Career progression suggestions - Industry insights

5. Skills Gap Analysis

Endpoint: `POST /api/ai-career/skills-gap`

Request:

```
{
  "target_role": "Data Scientist",
  "current_skills": ["Python", "SQL"]
}
```

Output: - Missing skills list - Proficiency levels needed - Learning resources

Implementation Flow

```
┌─────────────────────────────────────────────┐
│ 1. User Submits Query                       │
│    (Chat, CV, Job inquiry)                  │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Request to API Controller                │
│    AiCareerController@method                │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 3. Build User Context                       │
│    - University & major                     │
│    - GPA & graduation year                  │
│    - Skills & certifications                │
│    - Past applications                      │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 4. GeminiService Preparation                │
│    - Retrieve last 10 conversation messages │
│    - Build conversation history             │
│    - Format system prompt                   │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 5. API Call to Gemini                       │
│    POST {BASE_URL}/{MODEL}:generateContent  │
│    - API key in query param                 │
│    - Conversation context                   │
│    - Safety settings                        │
│    - Generation config                      │
└─────────────────────────────────────────────┘
       │
       │
       ▼
```

```
┌──────────────────────────────────────────────┐
│ 6. Gemini Processing                         │
│    - Natural language understanding          │
│    - Context-aware response generation       │
│    - Safety filtering                        │
└──────┬───────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────┐
│ 7. Response Parsing                          │
│    - Extract text from response              │
│    - Format for frontend                     │
└──────┬───────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────┐
│ 8. Save to Database                          │
│    - Table: ai_conversations                 │
│    - User message + AI response              │
│    - Timestamp                               │
└──────┬───────────────────────────────────────┘
       │
       ▼
┌──────────────────────────────────────────────┐
│ 9. Return to User                            │
│    JSON response with advice                 │
└──────────────────────────────────────────────┘
```

AI Configuration

Generation Settings:

```
[
    'temperature' => 0.7,        // Creativity level
    'maxOutputTokens' => 2048,   // Response length
    'topP' => 0.8,
    'topK' => 10
]
```

Safety Settings:

```
[
    'HARM_CATEGORY_HARASSMENT' => 'BLOCK_MEDIUM_AND_ABOVE',
    'HARM_CATEGORY_HATE_SPEECH' => 'BLOCK_MEDIUM_AND_ABOVE',
    'HARM_CATEGORY_SEXUALLY_EXPLICIT' => 'BLOCK_MEDIUM_AND_ABOVE',
    'HARM_CATEGORY_DANGEROUS_CONTENT' => 'BLOCK_MEDIUM_AND_ABOVE'
]
```

Key Integration Files

- `backend/app/Services/GeminiService.php` - Core AI service
- `backend/app/Http/Controllers/Api/AiCareerController.php` - API endpoints
- Database: `ai_conversations` table - Conversation history

Error Handling

```
try {
    $response = GeminiService::chat($message);
} catch (\Exception $e) {
    Log::error('Gemini API Error: ' . $e->getMessage());
    return response()->json([
        'error' => 'AI service temporarily unavailable'
    ], 503);
}
```

Cost Optimization

- **Model:** Using `gemini-1.5-flash` (cheaper than Pro)
- **Context:** Limited to last 10 messages
- **Caching:** Consider implementing response caching for common queries
- **Rate Limiting:** Implement user quotas

Cost Considerations

- **Pricing:** Pay-per-use (per 1M tokens)
- **Free Tier:** Limited daily requests
- **Recommendation:** Monitor token usage, implement quotas

---

4. JSearch Service - Job Aggregation

Purpose

Aggregates job postings from major platforms (LinkedIn, Indeed, Glassdoor, ZipRecruiter) via RapidAPI to supplement platform's local job listings.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 71-76)

Environment Variables:

```
JSEARCH_API_KEY=your_rapidapi_key
JSEARCH_DEFAULT_LOCATION=chicago
JSEARCH_DEFAULT_COUNTRY=us
JSEARCH_DEFAULT_SEARCH=developer
```

Documentation

- **Integration Guide:** `backend/docs/EXTERNAL_INTEGRATIONS.md`
- **Detailed Docs:** `JSEARCH_INTEGRATION.md`

API Integration

Endpoint: `https://jsearch.p.rapidapi.com/search`

Headers:

```
X-RapidAPI-Key: {JSEARCH_API_KEY}
X-RapidAPI-Host: jsearch.p.rapidapi.com
```

Implementation Flow

```
┌─────────────────────────────────────────────┐
│ 1. User Requests Jobs                       │
│    GET /api/jobs?job_source=external        │
│    or job_source=both                       │
└─────────────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. UniversityJobController Receives Request │
│    - Parse filters (keyword, location, etc) │
│    - Determine if external jobs needed      │
└─────────────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 3. JSearchService Called                    │
│    searchJobs($filters)                     │
```

```
└──────┬──────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 4. Build Query Parameters                   │
│    - query: keyword                         │
│    - location: city/country                 │
│    - employment_types: FULLTIME, PARTTIME   │
│    - remote_jobs_only: true/false           │
│    - date_posted: all/today/3days/week      │
└──────┬──────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 5. Check Cache                              │
│    Cache key: jsearch_{hash}                │
│    TTL: 1 hour                              │
└──────┬──────────────────────────────────────┘
       │
       │
       ├─ Cache Hit  ──────────────────────┐
       │                                   │
       └─ Cache Miss                       │
          │                                │
          ▼                                │
┌─────────────────────────────────────────────┐
│ 6. API Call to RapidAPI JSearch             │
│    GET /search with parameters              │
└──────┬──────────────────────────────────────┘
       │                                   │
       ▼                                   │
┌─────────────────────────────────────────────┐
│ 7. Response Transformation                  │
│    - Map to platform job format             │
│    - Prefix IDs with 'ext_'                 │
│    - Format salary ranges                   │
│    - Map experience levels                  │
│    - Sanitize UTF-8 encoding                │
└──────┬──────────────────────────────────────┘
       │                                   │
       ▼                                   │
┌─────────────────────────────────────────────┐
│ 8. Store in Cache                           │
└──────┬──────────────────────────────────────┘
       │                                   │
       ◄───────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 9. Combine with Local Jobs (if both)        │
│    - Merge arrays                           │
│    - Maintain separation by ID prefix       │
└──────┬──────────────────────────────────────┘
```

```
        │
        ▼
┌──────────────────────────────────────────────┐
│ 10. Return to Frontend                       │
│     JSON array of jobs                       │
└──────────────────────────────────────────────┘
```

Job Data Mapping

JSearch Response → Platform Format:

```
[
    'id' => 'ext_' . $job['job_id'],  // Prefix for external jobs
    'title' => $job['job_title'],
    'company' => $job['employer_name'],
    'location' => $job['job_city'] . ', ' . $job['job_country'],
    'description' => $job['job_description'],
    'salary_min' => $this->extractSalary($job, 'min'),
    'salary_max' => $this->extractSalary($job, 'max'),
    'employment_type' => $this->mapEmploymentType($job['job_employment_type']),
    'experience_level' => $this->mapExperienceLevel($job),
    'remote' => $job['job_is_remote'],
    'apply_url' => $job['job_apply_link'],
    'posted_at' => $job['job_posted_at_datetime_utc'],
    'source' => 'external'
]
```

Supported Filters

- **Keyword:** Job title, skills, company
- **Location:** City, state, country
- **Remote:** Remote jobs only filter
- **Employment Type:** Full-time, part-time, contract, internship
- **Date Posted:** All, today, 3 days, week, month
- **Experience Level:** Entry, mid, senior

Key Integration Files

- `backend/app/Services/JSearchService.php` - API client
- `backend/app/Http/Controllers/Api/UniversityJobController.php` - Controller
- `backend/app/Console/Commands/JSearchTest.php` - Testing CLI

- `backend/app/Console/Commands/JSearchClearCache.php` - Cache management

Error Handling

No API Key:

```
if (empty($apiKey)) {
    Log::warning('JSearch API key not configured');
    return [];  // Graceful degradation
}
```

API Errors:

```
catch (\Exception $e) {
    Log::error('JSearch API Error: ' . $e->getMessage());
    return [];  // Return empty array, don't break page
}
```

UTF-8 Issues:

```
// Sanitize to prevent encoding errors
$sanitized = mb_convert_encoding($text, 'UTF-8', 'UTF-8');
```

Cost Considerations

- **Free Tier:** 200 requests/month on RapidAPI
- **Basic Plan:** $9.99/month for 10,000 requests
- **Recommendation:** Implement 1-hour caching (already done)
- **Monitoring:** Track API usage via RapidAPI dashboard

---

5. Stripe Service - Primary Payment Gateway

Purpose

Primary payment processing solution for course enrollments, handling credit/debit card payments securely via Stripe.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 45-52)

Environment Variables:

```
STRIPE_KEY=pk_test_...                 # Publishable key
STRIPE_SECRET=sk_test_...              # Secret key
STRIPE_WEBHOOK_SECRET=whsec_...        # Webhook signing secret
STRIPE_WEBHOOK_TOLERANCE=300           # Signature tolerance (seconds)
```

Frontend Environment:

```
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_...
```

Composer Dependency:

```
"stripe/stripe-php": "^10.0"
```

Payment Flow

Phase 1: Create Payment Intent

```
┌─────────────────────────────────────────────┐
│ 1. Student Clicks "Enroll" on Course       │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Frontend Requests Payment Intent         │
│    POST /api/courses/{courseId}/payment/stripe │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 3. Backend Creates Payment Record           │
│    - Table: payments                        │
│    - Status: 'pending'                      │
│    - Amount: course.price (EGP)             │
│    - User & Course linked                   │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
```

```
│ 4. Create Stripe PaymentIntent                 │
│    Stripe\PaymentIntent::create([              │
│      'amount' => $amount * 100, // cents       │
│      'currency' => 'egp',                      │
│      'metadata' => [                           │
│        'payment_id' => $payment->id,           │
│        'course_id' => $courseId,               │
│        'user_id' => $userId                    │
│      ]                                         │
│    ])                                          │
└────────────────────────────────────────────────┘
       │
       │
       ▼
┌────────────────────────────────────────────────┐
│ 5. Return client_secret to Frontend            │
│    { client_secret: 'pi_xxx_secret_yyy' }      │
└────────────────────────────────────────────────┘
```

Phase 2: Frontend Payment Processing

```
┌────────────────────────────────────────────────┐
│ 6. Load Stripe.js on Frontend                  │
│    const stripe = Stripe(PUBLISHABLE_KEY)      │
└────────────────────────────────────────────────┘
       │
       │
       ▼
┌────────────────────────────────────────────────┐
│ 7. User Enters Card Details                    │
│    - Card number                               │
│    - Expiry date                               │
│    - CVC                                       │
│    - Billing details                           │
└────────────────────────────────────────────────┘
       │
       │
       ▼
┌────────────────────────────────────────────────┐
│ 8. Stripe.js Confirms Payment                  │
│    stripe.confirmCardPayment(client_secret, {│
│      payment_method: {                         │
│        card: cardElement,                      │
│        billing_details: {...}                  │
│      }                                         │
│    })                                          │
└────────────────────────────────────────────────┘
       │
       │
       ▼
┌────────────────────────────────────────────────┐
│ 9. Stripe Processes Payment                    │
│    - 3D Secure if required                     │
│    - Card authorization                        │
│    - Fraud detection                        │
└─────────────────────────────────────────────┘
```

Phase 3: Payment Confirmation

```
┌─────────────────────────────────────────────┐
│ 10. Frontend Confirms with Backend          │
│     POST /api/payment/confirm               │
│     { payment_intent_id: 'pi_xxx' }         │
└──────┬──────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 11. Backend Retrieves PaymentIntent         │
│     $intent = Stripe\PaymentIntent::retrieve│
│     ('pi_xxx')                              │
└──────┬──────────────────────────────────────┘
       │
       ├─ Status: succeeded ────────┐
       │                            │
       ├─ Status: failed ──────┐    │
       │                       │    │
       └─ Other ──────────┐    │    │
                          │    │    │
                          ▼    ▼    ▼
┌─────────────────────────────────────────────┐
│ 12. Update Payment Status                   │
│     - succeeded: 'completed'                │
│     - failed: 'failed'                      │
│     - other: no change                      │
└──────┬──────────────────────────────────────┘
       │ (if succeeded)
       ▼
┌─────────────────────────────────────────────┐
│ 13. Database Transaction Begins             │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 14. Create Course Enrollment                │
│     - Table: course_enrollments             │
│     - User + Course                         │
│     - Enrollment date                       │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 15. Update Course Statistics                │
│     - Increment students_count              │
│     - Update revenue                        │
```

```
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 16. Transaction Commits                     │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 17. Success Response to Frontend            │
│     { success: true, enrollment_id: 123 }   │
└─────────────────────────────────────────────┘
```

Phase 4: Webhook Backup (Async)

```
┌─────────────────────────────────────────────┐
│ Stripe Event Occurs                         │
│ (payment_intent.succeeded/failed, etc.)     │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ Stripe Sends Webhook                        │
│ POST /api/stripe/webhook                    │
│ Header: Stripe-Signature                    │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ Backend Verifies Signature                  │
│ Stripe\Webhook::constructEvent(             │
│   $payload, $signature, $secret             │
│ )                                           │
└─────────────────────────────────────────────┘
       │
       │
       ├─ Valid ─────────┐
       │                 │
       └─ Invalid ──┐    │
                    │    │
                    ▼    ▼
          [403 Error]    │
                         │
                         ▼
┌─────────────────────────────────────────────┐
│ Process Event Type                          │
│ - payment_intent.succeeded                  │
│ - payment_intent.payment_failed             │
│ - charge.refunded                           │
└─────────────────────────────────────────────┘
       │
       │
```

```
         ▼
┌─────────────────────────────────────────────┐
│ Update Payment & Enrollment (if needed)     │
│ - Idempotent operations                     │
│ - Handles edge cases                        │
└─────────────────────────────────────────────┘
```

Key Integration Files

- **Backend:**
  - `backend/app/Http/Controllers/Api/PaymentController.php` - Main payment logic
  - `backend/app/Http/Controllers/Api/StripeWebhookController.php` - Webhook handler
  - `backend/app/Models/Payment.php` - Payment model
  - `backend/app/Models/CourseEnrollment.php` - Enrollment model
- **Frontend:**
  - Payment form components
  - Stripe.js integration

Security Features

1. **Webhook Signature Verification:**

```
$event = \Stripe\Webhook::constructEvent(
    $payload, $signature, $webhookSecret
);
```

2. **Database Transactions:**
   - Ensures atomic operations (payment + enrollment)
   - Rollback on any failure
3. **Duplicate Prevention:**
   - Check if enrollment already exists
   - Idempotent webhook processing

4. **PCI Compliance:**
    - Card data never touches server
    - Handled entirely by Stripe.js

Error Handling

Payment Intent Creation:

```
try {
    $intent = \Stripe\PaymentIntent::create([...]);
} catch (\Stripe\Exception\ApiErrorException $e) {
    Log::error('Stripe API Error: ' . $e->getMessage());
    return response()->json(['error' => 'Payment failed'], 500);
}
```

Webhook Signature:

```
try {
    $event = \Stripe\Webhook::constructEvent(...);
} catch (\Stripe\Exception\SignatureVerificationException $e) {
    return response()->json(['error' => 'Invalid signature'], 403);
}
```

Testing

Test Cards: - Success: `4242 4242 4242 4242` - Decline: `4000 0000 0000 0002` - 3D Secure: `4000 0025 0000 3155`

Webhook Testing:

```
stripe listen --forward-to localhost:8000/api/stripe/webhook
stripe trigger payment_intent.succeeded
```

Cost Considerations

- **Transaction Fee:** 2.9% + EGP 2.50 per successful charge
- **No Monthly Fee:** Pay only for successful transactions
- **International Cards:** Additional 1% fee
- **Currency Conversion:** Automatic if needed

---

6. PayPal Service - Alternative Payment

Purpose

Secondary payment gateway for international students and users who prefer PayPal over credit cards.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 54-64)

Environment Variables:

```
PAYPAL_MODE=sandbox           # or 'live' for production

# Sandbox Credentials
PAYPAL_SANDBOX_CLIENT_ID=your_sandbox_client_id
PAYPAL_SANDBOX_SECRET=your_sandbox_secret

# Live Credentials
PAYPAL_LIVE_CLIENT_ID=your_live_client_id
PAYPAL_LIVE_SECRET=your_live_secret
```

Frontend Environment:

```
NEXT_PUBLIC_PAYPAL_CLIENT_ID=your_client_id
```

Composer Dependency:

```
"paypal/rest-api-sdk-php": "^1.14"
```

Payment Flow

```
┌─────────────────────────────────────────────┐
│ 1. Student Selects PayPal Payment Method    │
└─────────────────────────────────────────────┘
       │
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Frontend Requests PayPal Order           │
│    POST /api/courses/{courseId}/payment/paypal │
└─────────────────────────────────────────────┘
       │
```

```
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 3. Backend Currency Conversion              │
  │    EGP → USD                                │
  │    Rate: 1 USD ≈ 50 EGP (approximate)       │
  │    Example: 1000 EGP → 20 USD               │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 4. Create Payment Record                    │
  │    - Status: 'pending'                      │
  │    - Amount: original EGP                   │
  │    - Payment method: 'paypal'               │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 5. Return to Frontend                       │
  │    {                                        │
  │      payment_id: 123,                       │
  │      amount_usd: 20.00                      │
  │    }                                        │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 6. Frontend Loads PayPal SDK                │
  │    paypal.Buttons({                         │
  │      createOrder: () => { amount: 20.00 },  │
  │      onApprove: (data) => {...}             │
  │    })                                       │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 7. User Completes Payment on PayPal         │
  │    - Login to PayPal                        │
  │    - Review & confirm                       │
  │    - PayPal processes                       │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 8. PayPal Returns order_id                  │
  │    onApprove callback triggered             │
  └──────┬──────────────────────────────────────┘
         │
         ▼
  ┌─────────────────────────────────────────────┐
  │ 9. Frontend Confirms with Backend           │
```

```
│    POST /api/payment/paypal/confirm                   │
│    { payment_id: 123, order_id: 'xxx' }               │
└───────────────────────────────────────────────────────┘
          │
          │
          ▼
┌───────────────────────────────────────────────────────┐
│ 10. Backend Verification                              │
│     ⚠️ CURRENTLY SIMULATED                             │
│     Production should verify with PayPal API│
└───────────────────────────────────────────────────────┘
          │
          │
          ▼
┌───────────────────────────────────────────────────────┐
│ 11. Update Payment Status                             │
│     Status: 'completed'                               │
└───────────────────────────────────────────────────────┘
          │
          │
          ▼
┌───────────────────────────────────────────────────────┐
│ 12. Create Enrollment                                 │
│     - Grant course access                             │
│     - Update statistics                               │
└───────────────────────────────────────────────────────┘
          │
          │
          ▼
┌───────────────────────────────────────────────────────┐
│ 13. Success Response                                  │
└───────────────────────────────────────────────────────┘
```

Currency Conversion

Current Implementation:

```
$amountUSD = $course->price / 50;  // Simplified rate
```

Recommendation for Production:

```
// Use real-time exchange rate API
$rate = ExchangeRateService::getRate('EGP', 'USD');
$amountUSD = $course->price / $rate;
```

Important Security Note

⚠️ **CRITICAL:** The current PayPal verification is simulated. For production, implement proper order verification:

```
// Current (Simulated):
// Just marks payment as completed

// Required for Production:
use PayPal\Api\Payment;

$payment = Payment::get($orderId, $apiContext);
if ($payment->getState() === 'approved') {
    // Verify amount matches
    // Verify currency
    // Then complete enrollment
}
```

Key Integration Files

- `backend/app/Http/Controllers/Api/PaymentController.php`
    - `createPayPalPayment()` method
    - `confirmPayPalPayment()` method

Error Handling

Missing Order ID:

```
if (!$orderId) {
    return response()->json([
        'error' => 'PayPal order ID required'
    ], 400);
}
```

Verification Failure:

```
// TODO: Implement actual verification
catch (PayPalException $e) {
    Log::error('PayPal verification failed: ' . $e->getMessage());
    return response()->json(['error' => 'Payment verification failed'], 400)
}
```

Testing

Sandbox Mode: 1. Set `PAYPAL_MODE=sandbox` 2. Use sandbox client credentials 3. Test with PayPal sandbox accounts 4. Access sandbox: https://www.sandbox.paypal.com

Test Accounts: - Create at: https://developer.paypal.com/dashboard/accounts

Cost Considerations

- **Transaction Fee:** 4.4% + fixed fee (varies by country)
- **International Transactions:** Additional 1.5%
- **Currency Conversion:** PayPal's conversion rate applies
- **No Monthly Fee:** Pay only for transactions

Recommendations for Production

1. **Implement Proper Verification:**
    - Use PayPal SDK to verify order_id
    - Check payment status
    - Verify amount and currency
2. **Use Real Exchange Rates:**
    - Integrate with currency API
    - Update rates regularly
    - Display rate to user before payment
3. **Add Webhooks:**
    - Handle disputes and refunds
    - Automate payment status updates
4. **Enhance Error Handling:**
    - Detailed PayPal error messages
    - User-friendly error display

---

7. Jitsi Service - Live Video Conferencing

Purpose

FREE, open-source video conferencing solution for live class sessions, providing real-time audio/video communication without API costs.

Configuration

No API Keys Required!

Jitsi uses the public `meet.jit.si` infrastructure, which is: - ✅ Completely free - ✅ No signup required - ✅ No API limits - ✅ Production-ready

Alternative: Self-hosted Jitsi server (optional)

Implementation Flow

```
┌─────────────────────────────────────────────┐
│ 1. Teacher Creates Live Session             │
│    - POST /api/live/session/create          │
│    - Session ID generated                   │
│    - Scheduled time set                     │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Students Enroll in Course                │
│    - Can see scheduled sessions             │
│    - Receive notifications                  │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 3. User Joins Session                       │
│    - Click "Join" button                    │
│    - POST /api/live/session/{id}/join       │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 4. Backend Authorization Check              │
│    - Verify user is enrolled (students)     │
│    - Verify user owns session (teachers)    │
│    - Check session is active                │
└──────┬──────────────────────────────────────┘
       │
       ├─ Authorized ────────┐
       │                     │
       └─ Unauthorized ──┐   │
                         │   │
                         ▼   ▼
                    [403] Exit
                              │
                              ▼
┌─────────────────────────────────────────────┐
```

```
│ 5. Record Attendance                          │
│    - Table: session_attendances               │
│    - User + Session + Timestamp               │
└───────────────────────────────────────────────┘
       │
       │
       ▼
┌───────────────────────────────────────────────┐
│ 6. Redirect to Jitsi Page                     │
│    - /student/live-class/{sessionId}          │
│    - /teacher/live-class/{sessionId}          │
└───────────────────────────────────────────────┘
       │
       │
       ▼
┌───────────────────────────────────────────────┐
│ 7. Frontend Loads Jitsi External API          │
│    <script src="https://meet.jit.si/          │
│     external_api.js"></script>                │
└───────────────────────────────────────────────┘
       │
       │
       ▼
┌───────────────────────────────────────────────┐
│ 8. Initialize JitsiMeetExternalAPI            │
│    const api = new JitsiMeetExternalAPI(      │
│      'meet.jit.si', {                         │
│        roomName: 'EdvanceSession{id}',        │
│        width: '100%',                         │
│        height: '100%',                        │
│        parentNode: containerElement,          │
│        userInfo: {                            │
│          displayName: user.name,              │
│          email: user.email                    │
│        },                                     │
│        configOverwrite: {                     │
│          startWithAudioMuted: false,          │
│          startWithVideoMuted: false,          │
│          prejoinPageEnabled: false            │
│        }                                      │
│      }                                        │
│    )                                          │
└───────────────────────────────────────────────┘
       │
       │
       ▼
┌───────────────────────────────────────────────┐
│ 9. Jitsi Room Created/Joined                  │
│    - Room name: EdvanceSession{sessionId}     │
│    - All users with same sessionId join       │
│      the same room                            │
└───────────────────────────────────────────────┘
       │
       │
       ▼
┌───────────────────────────────────────────────┐
│ 10. Live Session Active                       │
│      - Video/audio streaming                     │
│      - Screen sharing                            │
│      - Chat                                      │
│      - Participant management                    │
└──────────────────────────────────────────────────┘
       │
       │
       ▼
┌──────────────────────────────────────────────────┐
│ 11. Teacher Can End Session                      │
│      - POST /api/live/session/{id}/end           │
│      - Status: 'ended'                           │
│      - Students disconnected                     │
└──────────────────────────────────────────────────┘
```

Room Naming Convention

Format: `EdvanceSession{sessionId}`

Example: - Session ID: 42 - Room Name: `EdvanceSession42`

Benefits: - Deterministic (same ID = same room) - No collisions between different sessions - Easy to identify in Jitsi logs

Jitsi Configuration Options

```
const config = {
  // Start with audio/video on
  startWithAudioMuted: false,
  startWithVideoMuted: false,

  // Skip pre-join page
  prejoinPageEnabled: false,

  // Branding (optional)
  DEFAULT_LOGO_URL: 'your-logo.png',
  DEFAULT_WELCOME_PAGE_LOGO_URL: 'your-logo.png',

  // Features
  toolbarButtons: [
    'microphone', 'camera', 'desktop', 'fullscreen',
    'fodeviceselection', 'hangup', 'profile',
    'chat', 'recording', 'livestreaming', 'etherpad',
    'sharedvideo', 'settings', 'raisehand',
    'videoquality', 'filmstrip', 'feedback',
    'stats', 'shortcuts', 'tileview', 'videobackgroundblur',
    'download', 'help', 'mute-everyone'
  ]
}
```

Features Available

✅ **Free Features:** - Unlimited participants - HD video - Screen sharing - Chat - Recording (to Dropbox) - YouTube live streaming - Raise hand - Reactions - Breakout rooms - End-to-end encryption

Key Integration Files

Frontend: - `frontend/app/student/live-class/[sessionId]/page.tsx` - `frontend/app/teacher/live-class/[sessionId]/page.tsx`

Backend: - `backend/app/Http/Controllers/Api/LiveStreamController.php` - `backend/app/Models/LiveSession.php` - `backend/app/Models/SessionAttendance.php`

Database Schema

live_sessions:

```
- id
- course_id
- teacher_id
- title
- scheduled_at
- status (scheduled/active/ended)
- created_at, updated_at
```

session_attendances:

```
- id
- session_id
- user_id
- joined_at
- created_at, updated_at
```

Event Listeners (Optional)

```
// User joined
api.addEventListener('participantJoined', (participant) => {
  console.log('Participant joined:', participant);
});

// User left
api.addEventListener('participantLeft', (participant) => {
  console.log('Participant left:', participant);
});

// Video conference joined
api.addEventListener('videoConferenceJoined', () => {
  console.log('You joined the conference');
});

// Video conference left
api.addEventListener('videoConferenceLeft', () => {
  console.log('You left the conference');
  // Redirect back to course page
});
```

Error Handling

Script Load Failure:

```
const script = document.createElement('script');
script.src = 'https://meet.jit.si/external_api.js';
script.onerror = () => {
  alert('Failed to load Jitsi. Please check your internet connection.');
};
document.head.appendChild(script);
```

Authorization Failure:

```
if (!$this->canJoinSession($user, $session)) {
    return response()->json([
        'error' => 'You are not authorized to join this session'
    ], 403);
}
```

Advantages Over Zoom/Teams

✅ **Free** - No costs at all ✅ **No API keys** - No signup required ✅ **Open source** - Full control ✅ **Privacy** - Can self-host ✅ **No time limits** - Unlimited session duration ✅ **No participant limits** - Scales well

Self-Hosting Option

For more control, you can deploy your own Jitsi server:

Benefits: - Full branding control - Custom features - Better privacy - No dependency on meet.jit.si

Requirements: - Ubuntu 20.04+ server - 4GB RAM minimum - Domain name - SSL certificate

Installation:

```
wget -qO - https://download.jitsi.org/jitsi-key.gpg.key | sudo apt-key add -
sudo sh -c "echo 'deb https://download.jitsi.org stable/' > /etc/apt/sources.list.d/jit
sudo apt update
sudo apt install jitsi-meet
```

Cost Considerations

- **Public Jitsi (meet.jit.si):** FREE ✅
- **Self-hosted:** Server costs only ($5-20/month for small scale)
- **8x8 (Commercial Jitsi):** Paid plans available with SLA

---

8. Agora Service - Alternative Video Streaming

Purpose

Cloud-based real-time audio/video streaming service, planned as an alternative to Jitsi with more advanced features and SDKs.

Configuration

Backend Configuration File: `backend/config/services.php` (lines 66-69)

Environment Variables:

```
AGORA_APP_ID=your_agora_app_id
AGORA_APP_CERTIFICATE=your_app_certificate
```

Current Implementation Status

⚠️ **Minimal Implementation:** - Basic token generation implemented - Currently simplified for development - **Jitsi is the active solution** for live sessions - Agora is available as a backup/ future option

Service Implementation

File: `backend/app/Services/AgoraService.php`

Current Token Generation:

```
public function generateToken($channelName, $uid = 0, $role = 1)
{
    $appId = config('services.agora.app_id');
    $appCertificate = config('services.agora.app_certificate');

    if (empty($appCertificate)) {
        // Development mode - return empty token
        return '';
    }

    // Simplified token generation
    // Production should use official Agora SDK
    $sessionData = [
        'channel' => $channelName,
        'uid' => $uid,
        'role' => $role,
        'timestamp' => time()
    ];

    return base64_encode(json_encode($sessionData));
}
```

Integration Points

Controller: `backend/app/Http/Controllers/Api/LiveStreamController.php`

```
// Currently not actively used
$token = AgoraService::generateToken($session->id);
```

Recommended Production Implementation

If you decide to use Agora, implement properly:

1. Install Official SDK:

```
composer require agora-php-sdk/agora-token-builder
```

2. Generate Proper Tokens:

```
use AgoraTokenBuilder\RtcTokenBuilder;

public function generateToken($channelName, $uid, $role)
{
    $appId = config('services.agora.app_id');
    $appCertificate = config('services.agora.app_certificate');
    $expireTimeInSeconds = 3600;
    $currentTimestamp = time();
    $privilegeExpiredTs = $currentTimestamp + $expireTimeInSeconds;

    return RtcTokenBuilder::buildTokenWithUid(
        $appId,
        $appCertificate,
        $channelName,
        $uid,
        $role,
        $privilegeExpiredTs
    );
}
```

3. Frontend Integration:

```
import AgoraRTC from "agora-rtc-sdk-ng"

const client = AgoraRTC.createClient({ mode: "rtc", codec: "vp8" })

await client.join(APP_ID, channelName, token, uid)
const localAudioTrack = await AgoraRTC.createMicrophoneAudioTrack()
const localVideoTrack = await AgoraRTC.createCameraVideoTrack()
await client.publish([localAudioTrack, localVideoTrack])
```

Agora Features

If fully implemented, Agora provides:

✅ **Advanced Features:** - Low latency (< 400ms globally) - Broadcast streaming - Cloud recording - AI noise suppression - Virtual backgrounds - Beauty filters - Screen sharing - Live transcription

Cost Considerations

Free Tier: - 10,000 minutes/month free - Suitable for testing

Paid Plans: - Audio: $0.99 per 1,000 minutes - HD Video: $3.99 per 1,000 minutes - Cloud Recording: Additional cost

Comparison: - More expensive than Jitsi (free) - Better SDKs and features - Enterprise support available

Why Jitsi is Currently Used

1. **Cost:** Jitsi is completely free
2. **Simplicity:** No API keys or SDK complexity
3. **Sufficient:** Meets current requirements
4. **Quick Integration:** Faster to implement

When to Switch to Agora

Consider Agora if you need: - Professional-grade streaming - Advanced features (AI noise suppression, beauty filters) - Better mobile app integration - Cloud recording with transcription - Guaranteed SLA - Better analytics

Key Integration Files

- `backend/app/Services/AgoraService.php` - Token generation
- `backend/app/Http/Controllers/Api/LiveStreamController.php` - Integration point

---

9. Pusher Service - Real-time Notifications

Purpose

WebSocket-based real-time communication for instant notifications, live updates, and event broadcasting across the platform.

Configuration

Expected Environment Variables:

```
PUSHER_APP_ID=your_app_id
PUSHER_APP_KEY=your_app_key
PUSHER_APP_SECRET=your_app_secret
PUSHER_APP_CLUSTER=mt1
PUSHER_SCHEME=https
PUSHER_HOST=
PUSHER_PORT=443
```

Composer Dependency:

```
"pusher/pusher-php-server": "^7.0"
```

Laravel Broadcasting Configuration

File: `config/broadcasting.php`

```
'pusher' => [
    'driver' => 'pusher',
    'key' => env('PUSHER_APP_KEY'),
    'secret' => env('PUSHER_APP_SECRET'),
    'app_id' => env('PUSHER_APP_ID'),
    'options' => [
        'cluster' => env('PUSHER_APP_CLUSTER'),
        'useTLS' => true,
        'encrypted' => true,
    ],
],
```

Typical Use Cases in ACE Platform

1. Live Session Notifications

Scenario: Teacher starts a live session

Backend Broadcasting:

```
use Illuminate\Support\Facades\Broadcast;

// When teacher starts session
event(new SessionStarted($session));
```

Event Class:

```
class SessionStarted implements ShouldBroadcast
{
    public $session;

    public function __construct($session)
    {
        $this->session = $session;
    }

    public function broadcastOn()
    {
        return new Channel('course.' . $this->session->course_id);
    }

    public function broadcastAs()
    {
        return 'session.started';
    }
}
```

Frontend Listening:

```
import Pusher from 'pusher-js'

const pusher = new Pusher(PUSHER_APP_KEY, {
  cluster: PUSHER_APP_CLUSTER
})

const channel = pusher.subscribe('course.' + courseId)

channel.bind('session.started', (data) => {
  // Show notification
  alert('Live class started! Join now.')
  // Auto-redirect option
  window.location.href = '/student/live-class/' + data.session.id
})
```

2. Job Application Status Updates

Backend:

```
event(new ApplicationStatusChanged($application));
```

Frontend:

```
const channel = pusher.subscribe('user.' + userId)
channel.bind('application.status.changed', (data) => {
  showNotification('Your application status: ' + data.status)
})
```

3. New Job Postings

Backend:

```
event(new JobPosted($job));
```

Frontend:

```
const channel = pusher.subscribe('university.' + universityId)
channel.bind('job.posted', (data) => {
  // Add to job list without refresh
  addJobToList(data.job)
})
```

4. Teacher Approval Notifications

Backend:

```
event(new TeacherApprovalStatus($teacher));
```

Frontend:

```
const channel = pusher.subscribe('private-user.' + userId)
channel.bind('teacher.approved', (data) => {
  showNotification('Your teacher account has been approved!')
  redirectToDashboard()
})
```

Private Channels (Authenticated)

Backend Route:

```
// routes/channels.php
Broadcast::channel('user.{id}', function ($user, $id) {
    return (int) $user->id === (int) $id;
});
```

Frontend:

```
// Subscribe to private channel
const channel = pusher.subscribe('private-user.' + userId)

// Pusher will automatically hit /broadcasting/auth
// to verify user can access this channel
```

Presence Channels (Who's Online)

Use Case: Show active participants in live session

Backend:

```
Broadcast::channel('presence-session.{id}', function ($user, $sessionId) {
    if ($user->canJoinSession($sessionId)) {
        return [
            'id' => $user->id,
            'name' => $user->name,
            'role' => $user->role
        ];
    }
});
```

Frontend:

```
const channel = pusher.subscribe('presence-session.' + sessionId)

// Member added
channel.bind('pusher:member_added', (member) => {
  console.log(member.info.name + ' joined')
  updateParticipantsList()
})

// Member removed
channel.bind('pusher:member_removed', (member) => {
  console.log(member.info.name + ' left')
  updateParticipantsList()
})

// Get all current members
const members = channel.members
members.each((member) => {
  console.log(member.info.name)
})
```

Implementation Flow

```
┌─────────────────────────────────────────────┐
│ 1. Event Occurs (Session starts, etc.)     │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 2. Laravel Dispatches Event                 │
│    event(new SessionStarted($session))      │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 3. Event Implements ShouldBroadcast         │
│    - Define channel(s)                      │
│    - Define event name                      │
│    - Define data payload                    │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 4. Laravel Broadcasting Sends to Pusher     │
│    POST https://api-{cluster}.pusher.com    │
│    - App ID authentication                  │
│    - Channel name                           │
│    - Event name                             │
│    - Data payload                           │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 5. Pusher Receives & Broadcasts             │
│    - Validates credentials                  │
│    - Broadcasts to subscribed clients       │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 6. Frontend Clients Receive Event           │
│    channel.bind('session.started', ...)     │
└──────┬──────────────────────────────────────┘
       │
       ▼
┌─────────────────────────────────────────────┐
│ 7. UI Updates in Real-time                  │
│    - Show notification                      │
│    - Update UI                              │
│    - Trigger actions                          │
└───────────────────────────────────────────────┘
```

Error Handling

Connection Errors:

```
pusher.connection.bind('error', (err) => {
  if (err.error.data.code === 4004) {
    console.error('Over connection limit')
  }
})

pusher.connection.bind('state_change', (states) => {
  console.log('State changed:', states.current)
  // connected, connecting, disconnected, etc.
})
```

Failed Subscription:

```
channel.bind('pusher:subscription_error', (status) => {
  console.error('Subscription failed:', status)
})
```

Best Practices

1. **Channel Naming:**
    - Public: `job-board`, `course.{id}`
    - Private: `private-user.{id}`
    - Presence: `presence-session.{id}`
2. **Data Size:**
    - Keep payloads small (< 10KB)
    - Send IDs, not full objects
    - Fetch details via API if needed
3. **Connection Management:**
    - Disconnect when page unloads
    - Unsubscribe from channels when done
    - Handle reconnection gracefully

4. **Security:**
   - Use private channels for sensitive data
   - Validate authorization in channel routes
   - Never expose secrets in frontend

Cost Considerations

Free Tier: - 200,000 messages/day - 100 max connections - Suitable for development and small apps

Paid Plans: - Channels: $49/month (500 max connections) - Business: $299/month (2,000 max connections) - Enterprise: Custom pricing

Alternatives: - Laravel WebSockets (self-hosted, free) - Ably (similar pricing) - Socket.io (self-hosted)

Alternative: Laravel WebSockets

For cost savings, consider self-hosted Laravel WebSockets:

Installation:

```
composer require beyondcode/laravel-websockets
```

Benefits: - FREE (no per-message costs) - Drop-in Pusher replacement - Same API - Full control

Drawbacks: - Requires server management - Scaling more complex - No automatic failover

---

Summary & Comparison

Service Overview Table

Service	Category	Status	Cost Model	Purpose
DiDit	Identity Verification	✅ Active	Pay-per-verification	Student/teacher ID verification
MailGun	Email Delivery	✅ Active		Transactional emails

Service	Category	Status	Cost Model	Purpose
			Freemium + Pay-per-email	
Gemini	AI/ML	✅ Active	Pay-per-token	Career guidance & CV analysis
JSearch	Job API	✅ Active	Freemium + Subscription	External job listings
Stripe	Payment Gateway	✅ Active	Transaction fee (2.9% + fee)	Primary payment processing
PayPal	Payment Gateway	⚠️ Needs work	Transaction fee (~4.4%)	Alternative payment
Jitsi	Video Conferencing	✅ Active	FREE	Live class sessions
Agora	Video Streaming	⏸️ Minimal	Freemium + Pay-per-minute	Alternative to Jitsi
Pusher	WebSockets	✅ Active	Freemium + Subscription	Real-time notifications

Monthly Cost Estimate (Approximate)

Scenario: 1,000 active users, 100 courses, 50 live sessions/month

Service	Free Tier	Estimated Cost
DiDit	No	~$200-500 (ID verifications)
MailGun	5,000/month	FREE (under limit)
Gemini	Limited	~$20-50 (based on usage)
JSearch	200 requests/month	FREE or $9.99
Stripe	No	~$300 (on $10K revenue)
PayPal	No	~$440 (on $10K revenue)
Jitsi	Unlimited	FREE ✅

Service	Free Tier	Estimated Cost
Agora	10,000 min/month	FREE (under limit)
Pusher	200K messages/day	FREE or $49
TOTAL	-	**~$570-1,000/month**

Architecture Patterns Used

1. Webhook Pattern

- **Used by:** DiDit, Stripe
- **Purpose:** Async event notifications
- **Security:** Signature verification

2. API Gateway Pattern

- **Used by:** All services
- **Purpose:** Centralized API access
- **Implementation:** Laravel controllers

3. Service Layer Pattern

- **Files:** `app/Services/`
- **Purpose:** Encapsulate external API logic
- **Services:** GeminiService, JSearchService, AgoraService

4. Observer Pattern

- **Used by:** Laravel Events + Pusher
- **Purpose:** Real-time notifications
- **Implementation:** Event broadcasting

5. Strategy Pattern

- **Used by:** Payment methods (Stripe vs PayPal)
- **Purpose:** Interchangeable payment processors

Configuration Management

All services use environment variables for: - ✅ **Security:** Secrets not in code - ✅ **Flexibility:** Easy environment switching - ✅ **Deployment:** Different configs for dev/staging/prod

Files: - `backend/.env` - Secret values (not committed) - `backend/.env.example` - Template - `backend/config/services.php` - Service configurations - `frontend/.env.local` - Frontend secrets

Error Handling Strategy

All integrations implement: 1. **Try-Catch Blocks:** Graceful error handling 2. **Logging:** Comprehensive error logs 3. **Graceful Degradation:** Continue functioning when service unavailable 4. **User-Friendly Messages:** Clear error communication

Security Best Practices

✅ **API Keys:** - Stored in environment variables - Never committed to Git - Rotated regularly

✅ **Webhook Verification:** - DiDit: HMAC-SHA256 signatures - Stripe: Signature verification - Prevents spoofed webhooks

✅ **HTTPS Only:** - All API calls over HTTPS - SSL verification enabled (except dev mode)

✅ **Input Validation:** - All user inputs sanitized - SQL injection prevention - XSS protection

✅ **Rate Limiting:** - Prevent API abuse - Protect from DDoS

Testing Recommendations

1. Unit Tests:

```
// Test GeminiService
public function test_gemini_generates_response()
{
    $response = GeminiService::chat('Hello');
    $this->assertNotEmpty($response);
}
```

2. Integration Tests:

```
// Test Stripe payment flow
public function test_payment_creates_enrollment()
{
    $response = $this->post('/api/payment/confirm', [
        'payment_intent_id' => 'pi_test_123'
    ]);
    $this->assertDatabaseHas('course_enrollments', [
        'user_id' => $this->user->id
    ]);
}
```

3. Webhook Testing:

```
# Stripe CLI
stripe listen --forward-to localhost:8000/api/stripe/webhook
stripe trigger payment_intent.succeeded
```

Monitoring & Observability

Recommended Tools: 1. **Laravel Telescope** - Request/query debugging 2. **Log Management** - Centralized logs (e.g., Papertrail) 3. **Uptime Monitoring** - Ping webhooks (e.g., UptimeRobot) 4. **Error Tracking** - Sentry or Bugsnag 5. **API Analytics** - Track usage and costs

Scalability Considerations

Current Architecture: - ✅ Stateless API (scales horizontally) - ✅ External services handle heavy lifting - ✅ Caching implemented (JSearch: 1 hour)

Future Improvements: 1. **Queue System:** Laravel Queues for async processing 2. **Redis Caching:** Faster than file cache 3. **CDN:** For static assets 4. **Database Read Replicas:** For high-traffic reads 5. **Load Balancer:** Distribute traffic

---

Security Considerations

API Key Management

Storage:

```
# Never commit .env file
echo ".env" >> .gitignore

# Use .env.example as template
cp .env.example .env
```

Key Rotation: - Rotate keys every 90 days - Use separate keys for dev/staging/prod - Revoke compromised keys immediately

Webhook Security

Signature Verification:

```
// Always verify webhook signatures
$signature = $request->header('Stripe-Signature');
$event = \Stripe\Webhook::constructEvent(
    $payload, $signature, $webhookSecret
);
```

IP Whitelisting (Optional):

```
// In middleware
$allowedIPs = ['209.85.128.0/17']; // Stripe IPs
if (!in_array($request->ip(), $allowedIPs)) {
    abort(403);
}
```

Data Privacy

GDPR Compliance: - User consent for data processing - Right to data deletion - Data portability

Payment Data: - PCI DSS compliance via Stripe/PayPal - Never store card numbers - Tokenization for recurring payments

Rate Limiting

Implementation:

```
// routes/api.php
Route::middleware('throttle:60,1')->group(function () {
    // 60 requests per minute per user
});
```

Per-Service Limits: - Gemini AI: 10 requests/minute per user - JSearch: Managed by cache - Payment: Extra strict limits

Audit Logging

Track Critical Events: - Payment transactions - ID verifications - Admin approvals - Account changes

Implementation:

```
Log::info('Payment completed', [
    'user_id' => $user->id,
    'amount' => $amount,
    'payment_id' => $payment->id,
    'ip_address' => $request->ip()
]);
```

Appendix: Quick Reference

Environment Variables Checklist

```
# DiDit
DIDIT_API_KEY=
DIDIT_WEBHOOK_SECRET=
DIDIT_WORKFLOW_ID=

# MailGun
MAILGUN_DOMAIN=
MAILGUN_SECRET=
MAILGUN_ENDPOINT=

# Gemini
GEMINI_API_KEY=
GEMINI_MODEL=gemini-1.5-flash

# JSearch
JSEARCH_API_KEY=

# Stripe
STRIPE_KEY=
STRIPE_SECRET=
STRIPE_WEBHOOK_SECRET=

# PayPal
PAYPAL_MODE=sandbox
PAYPAL_SANDBOX_CLIENT_ID=
PAYPAL_SANDBOX_SECRET=
PAYPAL_LIVE_CLIENT_ID=
PAYPAL_LIVE_SECRET=

# Agora (optional)
AGORA_APP_ID=
AGORA_APP_CERTIFICATE=

# Pusher
PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=
```

Common Commands

```
# Clear JSearch cache
php artisan jsearch:clear-cache

# Test JSearch integration
php artisan jsearch:test "developer" "chicago"

# Test Stripe webhook
stripe listen --forward-to localhost:8000/api/stripe/webhook

# View logs
tail -f storage/logs/laravel.log

# Clear all caches
php artisan cache:clear
php artisan config:clear
php artisan route:clear
```

Support & Documentation Links

Service	Documentation	Support
DiDit	https://docs.didit.me	support@didit.me
MailGun	https://documentation.mailgun.com	https://help.mailgun.com
Gemini	https://ai.google.dev/docs	https://ai.google.dev/support
JSearch	https://rapidapi.com/letscrape-6bRBa3QguO5/api/jsearch	RapidAPI Support
Stripe	https://stripe.com/docs	https://support.stripe.com
PayPal	https://developer.paypal.com/docs	PayPal Developer Support

Service	Documentation	Support
Jitsi	https://jitsi.github.io/handbook	Community Forum
Agora	https://docs.agora.io	support@agora.io
Pusher	https://pusher.com/docs	support@pusher.com

End of Document

For questions or updates to this guide, contact the development team.