Path2Hire Application - Complete Developer Documentation

Table of Contents

1. Overview
2. Architecture
3. Project Structure
4. Technology Stack
5. Setup & Installation
6. Application Flow & Pipelines
7. Core Modules
8. API Routes & Endpoints
9. Authentication System
10. Payment Integration
11. Assessment System
12. Report Generation
13. Data Storage
14. Deployment
15. Configuration
16. Troubleshooting

Overview

Path2Hire is a career assessment and training platform that helps users: - Take personalized career assessments - Receive detailed career reports and recommendations - Access job listings - Apply for training programs - Manage profiles and career applications

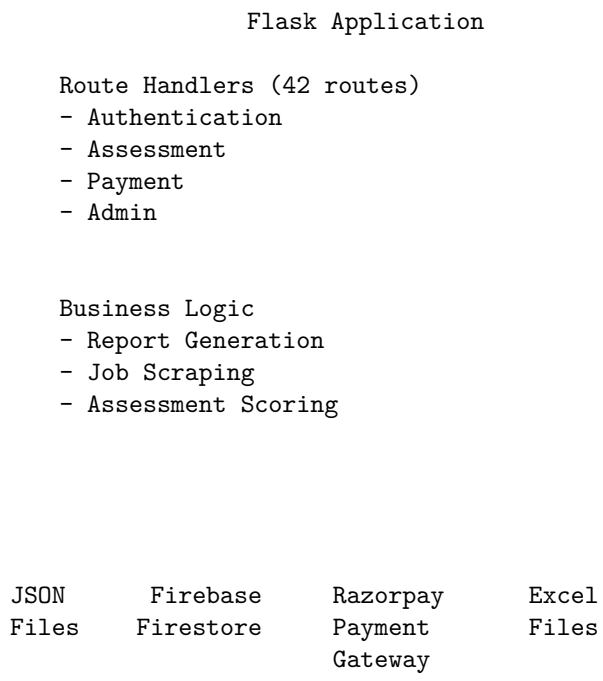
Application Type: Flask-based web application
Primary Language: Python 3.x
Frontend: HTML/CSS/JavaScript (Tailwind CSS)
Database: JSON files (local) + Firebase Firestore (cloud)

Architecture

High-Level Architecture

```
                    Client Browser
(HTML/CSS/JS - Tailwind CSS, Font Awesome Icons)


                      HTTP/HTTPS
```



Request Flow

1. **User Request** → Flask Route Handler
2. **Authentication Check** → Session validation
3. **Business Logic** → Process request
4. **Data Access** → Read/Write to JSON files or Firestore
5. **Response** → Render template or return JSON

Project Structure

```
path2hire-working-10-24-25/

  app.py                          # Main Flask application (1909 lines)
  career_report_generator.py       # Report generation logic
  job_scraper.py                  # Job scraping functionality
  check_excel.py                  # Excel validation utility
  requirements.txt                # Python dependencies
  Procfile                        # Production deployment config

  templates/                      # Flask HTML templates
```

```
    assessment.html            # Assessment interface
    results.html               # Results display
    payment.html               # Payment page
    profile.html                # User profile
    trainer.html               # Trainer application form
    jobs.html                  # Job listings

site/                           # Static website files
    index.html                 # Homepage
    login.html                 # Login/signup page
    about.html                 # About page
    contact.html               # Contact page
    programs/                  # Training program pages
    assets/                    # CSS, JS, images
        css/custom.css
        js/

data/                           # Application data
    careers/                   # Career applications
        careers.xlsx
        [Resume/CV files]
    contacts/                  # Contact form submissions
        contacts.xlsx
    trainers/                  # Trainer data

attempts/                       # Assessment attempts (JSON files)
payments/                       # Payment records
    payments.json

users.json                      # User accounts
firebase-service-account.json  # Firebase credentials
Sample Assessment questions.xlsx  # Assessment questions source
```

Technology Stack

Backend

- **Flask** - Web framework
- **Python 3.x** - Programming language
- **Firebase Admin SDK** - Authentication & Firestore
- **Razorpay** - Payment gateway
- **Pandas** - Data processing
- **ReportLab** - PDF generation
- **BeautifulSoup4** - Web scraping
- **Werkzeug** - Password hashing

Frontend

- **HTML5/CSS3** - Markup & styling
- **Tailwind CSS** - Utility-first CSS framework
- **JavaScript** - Client-side logic
- **Font Awesome** - Icons
- **Firebase Client SDK** - Client-side auth

Data Storage

- **JSON Files** - Local file storage
- **Excel Files** - Data export/import (openpyxl)
- **Firebase Firestore** - Cloud database

Setup & Installation

Prerequisites

- Python 3.8 or higher
- pip (Python package manager)
- Firebase project with service account
- Razorpay account

Installation Steps

1. **Clone/Download the project**

   ```
   cd path2hire-working-10-24-25
   ```

2. **Create virtual environment** (recommended)

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

3. **Install dependencies**

   ```
   pip install -r requirements.txt
   ```

4. **Configure Firebase**
   - Place `firebase-service-account.json` in the root directory
   - This file contains Firebase credentials for server-side auth

5. **Configure Razorpay**
   - Update `RAZORPAY_KEY_ID` and `RAZORPAY_KEY_SECRET` in `app.py` (lines 236-237)
   - Currently using live keys (consider moving to environment variables)

6. **Set Secret Key**

- **IMPORTANT**: Change `app.secret_key` in `app.py` (line 40)
- Current value: `'unicorn-secret-please-change'`
- Use a strong random secret for production

7. **Create required directories**
   - Directories are auto-created, but verify:
     - `attempts/` - For assessment attempts
     - `data/careers/` - For career applications
     - `data/contacts/` - For contact submissions
     - `data/trainers/` - For trainer data
     - `payments/` - For payment records
8. **Run the application**

   ```
   python app.py
   ```

   Application runs on `http://0.0.0.0:5000`

Development vs Production

Development: - Uses Flask development server - Debug mode: OFF (line 41) - Runs on port 5000

Production: - Use Gunicorn (included in requirements.txt) - Use the `Procfile` for deployment platforms (Heroku, etc.) - Command: `gunicorn app:app`

Application Flow & Pipelines

1. User Registration & Authentication Pipeline

```
User Visits Login Page
    ↓
Choose: Email/Password OR Google Sign-In
    ↓

  Option 1: Email/Password
    - Sign up (creates account)
    - Sign in (validates credentials)

    OR

  Option 2: Google Sign-In (Firebase)
    - Click "Sign in with Google"
    - Firebase popup/redirect
    - Get ID token from Firebase
    - Send token to /firebase-login
    - Server verifies token
```

```
   - Creates/updates user in users.json
   - Sets session

    ↓
Session Created → Redirect to Next URL or /assessment
```

Key Files: - `site/login.html` - Login/signup UI - `app.py` lines 1082-1134 - Auth routes - `app.py` lines 70-115 - Firebase auth

Session Management: - Sessions stored server-side - Cookie-based with HttpOnly flag - Session timeout: None (managed manually)

2. Payment Pipeline

```
User Needs Assessment Access
    ↓
Redirect to /payment
    ↓
Check if already paid (has_user_paid())
    ↓

 Payment Flow:
 1. User clicks "Pay Now"
 2. Frontend calls /create_order
 3. Server creates Razorpay order
 4. Frontend opens Razorpay checkout
 5. User completes payment
 6. Razorpay calls callback
 7. Frontend calls /verify_payment
 8. Server verifies signature
 9. Payment recorded in payments.json

    ↓
Payment Verified → Redirect to /assessment
```

Key Files: - `templates/payment.html` - Payment UI - `app.py` lines 467-572 - Payment routes - `payments/payments.json` - Payment records

Payment Amount: 199.00 (set in `ASSESSMENT_PRICE`)

3. Assessment Pipeline

```
User Accesses /assessment
    ↓
Check: Logged in? → Yes
Check: Paid? → Yes
    ↓
Create/Resume Assessment Attempt
```

```
    ↓

  Assessment Flow:
  1. Load questions from Excel
  2. Randomize questions (100 questions)
  3. Create attempt JSON file
  4. Start timer (30 minutes)
  5. Display questions one-by-one
     (route: /assessment/<idx>)
  6. User answers each question
     (route: /answer POST)
  7. Save answers to attempt file
  8. Continue until all answered
  9. Submit assessment
     (route: /submit)
  10. Calculate scores
  11. Generate attributes
  12. Save results

    ↓
Redirect to /results
```

Key Files: - `templates/assessment.html` - Assessment UI - `app.py` lines 1217-1396 - Assessment routes - `attempts/*.json` - Assessment attempt files - `Sample Assessment questions.xlsx` - Question bank

Assessment Details: - 100 questions total - 10 questions per category (10 categories) - 30-minute time limit - One question per page - Auto-save answers

4. Results & Report Generation Pipeline

```
Assessment Submitted
    ↓
Calculate Scores
    ↓

  Scoring System:
  - Count answers for each category:
    * FAR (Financial/Analytical Reasoning)
    * BM (Business/Market Acumen)
    * CRM (Customer Relationship/Comm)
    * MO (Motivation/Operational)
  - Calculate percentages
  - Generate 10 attribute scores
  - Determine strongest category

    ↓
```

```
Generate Report Context
    ↓

  Report Generation:
  1. Map scores to career track
  2. Generate SWOT analysis
  3. Create career recommendations
  4. Build markdown report

    ↓
Display Results (/results)
    ↓
User Can Download PDF (/download_career_blueprint)
```

Key Files: - `templates/results.html` - Results display - `career_report_generator.py` - Report logic - `app.py` lines 1426-1720 - Results & download routes

5. Contact Form Pipeline

```
User Fills Contact Form
    ↓
Submit to /contact (POST)
    ↓

  Data Storage:
  1. Save to Firebase Firestore
     (collection: 'contacts')
  2. Save to Excel file
     (data/contacts/contacts.xlsx)

    ↓
Return Success Response
```

6. Career Application Pipeline

```
User Fills Career Form
    ↓
Upload Resume & Cover Letter
    ↓
Submit to /career (POST)
    ↓

  Data Storage:
  1. Save files to data/careers/
     - RESUME_<timestamp>_<filename>
     - CL_<timestamp>_<filename>
  2. Save to Firebase Firestore
```

```
    (collection: 'career_applications')
  3. Save to Excel file
    (data/careers/careers.xlsx)

    ↓
Return Success Response
```

Core Modules

1. `app.py` (Main Application)

Purpose: Central Flask application with all routes and core logic

Key Sections: - **Lines 50-115**: Firebase Authentication setup - **Lines 178-163**: Contact form handler - **Lines 233-239**: Razorpay configuration - **Lines 250-328**: Utility functions (load_users, save_users, etc.) - **Lines 467-572**: Payment routes - **Lines 670-893**: Trainer application routes - **Lines 896-901**: Career application handler - **Lines 988-1072**: Admin dashboard routes - **Lines 1078-1134**: Authentication routes - **Lines 1146-1211**: Profile routes - **Lines 1217-1396**: Assessment routes - **Lines 1426-1720**: Results & report download routes - **Lines 1737-1842**: PDF report generation (download_report) - **Lines 1869-1905**: Jobs & static file routes

Key Functions: - `load_users()` - Load user data from JSON - `save_users()` - Save user data to JSON - `has_user_paid()` - Check payment status - `load_questions()` - Load assessment questions from Excel - `calculate_attribute_scores()` - Calculate attribute scores

2. `career_report_generator.py`

Purpose: Generate career assessment reports

Key Functions: - `map_assessment_to_report(scores)` - Map scores to career track - `calculate_attribute_scores(assessment_scores)` - Calculate 10 attribute scores - `generate_career_blueprint_report(user_name, scores, attributes)` - Generate full report - `generate_swot_analysis()` - Generate SWOT analysis - `get_career_recommendations()` - Get career path recommendations - `expand_abbreviations()` - Replace abbreviations with full names

Attribute Mapping: - Accounting Knowledge - Quantitative & Math Skill - Analytical & Critical Thinking - Attention to Detail - Financial Concepts - Compliance & Ethics - Business & Economic Acumen - Communication Skills - Tech & Tool Familiarity - Personality Preferences

3. `job_scraper.py`

Purpose: Scrape job listings from Google Jobs

Key Class: - `GoogleJobsScraper` - Scrapes Google Jobs search results

Key Function: - `get_latest_jobs(limit=50)` - Returns list of job dictionaries

Job Data Structure:

```
{
    'title': str,
    'company': str,
    'location': str,
    'description': str,
    'url': str,
    'posted': str
}
```

API Routes & Endpoints

Authentication Routes

Route	Method	Description	Auth Required
`/`	GET	Homepage	No
`/login`	GET, POST	Login page & authentication	No
`/signup`	POST	User registration	No
`/logout`	GET	Logout user	Yes
`/firebase-login`	POST	Firebase authentication	No

Assessment Routes

Route	Method	Description	Auth Required
`/assessment`	GET	Start assessment	Yes + Paid
`/assessment/<idx>`	GET	Display question at index	Yes
`/answer`	POST	Save answer	Yes
`/submit`	GET, POST	Submit assessment	Yes
`/results`	GET	View results	Yes

Payment Routes

Route	Method	Description	Auth Required
/payment	GET	Payment page	Yes
/create_order	POST	Create Razorpay order	Yes
/verify_payment	POST	Verify payment signature	Yes
/checkpoint	GET	Payment bypass (testing)	Yes

Profile & User Routes

Route	Method	Description	Auth Required
/profile	GET	User profile page	Yes
/profile/update	POST	Update profile	Yes

Trainer Routes

Route	Method	Description	Auth Required
/trainer	GET	Trainer application form	Yes
/trainer/update/personal	POST	Update personal info	Yes
/trainer/update/identification	POST	Upload ID documents	Yes
/trainer/update/banking	POST	Update banking details	Yes
/trainer/update/qualifications	POST	Update qualifications	Yes
/trainer/update/employment	POST	Update employment history	Yes
/trainer/update/trainings	POST	Update training courses	Yes

Admin Routes

Route	Method	Description	Auth Required
/admin	GET	Admin dashboard	Admin only
/admin/dashboard	GET	Admin dashboard	Admin only
/admin/contacts	GET	View contact submissions	Admin only
/admin/careers	GET	View career applications	Admin only
/admin/contact/<id>/status	POST	Update contact status	Admin only
/admin/career/<id>/status	POST	Update career status	Admin only

Download Routes

Route	Method	Description	Auth Required
`/download_career_blueprint`	GET	Download PDF report	Yes
`/download_report`	GET	Download assessment PDF	Yes
`/download/careers`	GET	Download careers zip	Admin
`/download/contacts`	GET	Download contacts zip	Admin
`/download/careers/delete`	POST	Delete careers data	Admin
`/download/contacts/delete`	POST	Delete contacts data	Admin

Form Submission Routes

Route	Method	Description	Auth Required
`/contact`	POST	Submit contact form	No
`/career`	POST	Submit career application	No

Jobs Routes

Route	Method	Description	Auth Required
`/jobs`	GET	Jobs listing page	No
`/api/jobs`	GET	Get jobs JSON API	No

Utility Routes

Route	Method	Description	Auth Required
`/health`	GET	Health check endpoint	No
`/<path:filename>`	GET	Serve static files	No

Authentication System

Authentication Methods

1. **Email/Password Authentication**
   - Stores hashed passwords using Werkzeug
   - Password hash: `scrypt:32768:8:1$...`
   - User data stored in `users.json`
2. **Firebase Google Sign-In**
   - Uses Firebase Authentication SDK
   - Client-side: `site/login.html` (Firebase JS SDK)

- Server-side: `/firebase-login` route
- Verifies ID tokens with Firebase Admin SDK
- Handles clock skew errors (up to 60 seconds tolerance)

Session Management

- **Session Storage**: Server-side (Flask sessions)
- **Session Cookie**: HttpOnly, SameSite=Lax
- **Session Key**: `logged_in` (boolean)
- **User Data**: `session['user'] = {'email': str, 'name': str}`

Admin Access

- **Admin Email**: `'admin'` (hardcoded)
- **Admin Password**: `'admin'` (hardcoded)
- **Security Note**: Change admin credentials in production!

Payment Integration

Razorpay Configuration

Current Setup: - **Key ID**: `rzp_live_RRLzuRNwQiqFcR` (Live key) - **Key Secret**: `1Fct2RgdkxW97AWMTTRYsunC` (Live secret) - **Amount**: 199.00 (`ASSESSMENT_PRICE`)

Payment Flow: 1. Frontend calls `/create_order` 2. Server creates Razorpay order 3. Frontend opens Razorpay checkout 4. User pays 5. Razorpay redirects back 6. Frontend calls `/verify_payment` 7. Server verifies payment signature 8. Payment saved to `payments/payments.json`

Payment Data Structure:

```
{
  "user_email": [
    {
      "payment_id": "pay_...",
      "order_id": "order_...",
      "amount": 19900,
      "currency": "INR",
      "status": "captured",
      "timestamp": "2025-...",
      "user_email": "user@example.com"
    }
  ]
}
```

Receipt Format: - Format: `assess_{email_prefix}_{short_uuid}` - Max length: 40 characters (Razorpay requirement) - Fallback: Uses timestamp if too long

---

Assessment System

Question Structure

Source: `Sample Assessment questions.xlsx`

Excel Columns: - `No` - Question number - `Scenario` - Question text - `Option A`, `Option B`, `Option C`, `Option D` - Answer options - `Categories/Attributes` - Question category

Categories (10 total): 1. Accounting Knowledge 2. Attention to Detail 3. Business & Economic Acumen 4. Communication Skills 5. Compliance & Ethics 6. Financial Concepts Skill 7. Personality Preference 8. Problem Solving Skills 9. Quantitative & Math Skill 10. Tech & Tool Familiarity

Answer Codes: - `FAR` - Financial/Analytical Reasoning - `BM` - Business/Market Acumen - `CRM` - Customer Relationship/Communication - `MO` - Motivation/Operational Orientation

Assessment Process

1. **Question Loading** (`load_questions()`)
   - Loads from Excel file
   - Groups by category
   - Selects 10 questions per category (100 total)
   - Randomizes questions and options
2. **Attempt Creation**
   - Creates unique attempt ID (UUID)
   - Saves to `attempts/{attempt_id}.json`
   - Includes: user, start time, questions, answers (empty)
3. **Answer Submission**
   - User answers one question at a time
   - Answers saved to attempt file
   - Format: `{"question_no": "answer_code"}`
4. **Scoring** (`/submit` route)
   - Counts answers per category (FAR, BM, CRM, MO)
   - Calculates percentages
   - Generates 10 attribute scores
   - Determines strongest category
   - Saves results to attempt file

Time Management

- **Time Limit**: 30 minutes per attempt
- **Timeout Check**: On assessment resume
- **Expired Attempts**: Auto-deleted and session cleared

Report Generation

Report Types

1. **Career Blueprint Report** (`/download_career_blueprint`)
   - Full markdown report converted to PDF
   - Includes: attributes, SWOT, recommendations, next steps
   - Generated by `generate_career_blueprint_report()`
2. **Assessment Report** (`/download_report`)
   - Basic PDF report
   - Includes: scores, strengths, weaknesses, narratives
   - Generated using ReportLab

PDF Generation

Technology: ReportLab

Process: 1. Generate markdown content 2. Parse markdown to PDF elements 3. Apply styling (headers, tables, lists) 4. Add footer with contact info 5. Build PDF document

Footer Information: - Path2Hire contact details - Copyright notice - Appears on every page

Styling: - Custom colors for headings - Professional table layouts - Proper spacing and formatting

---

Data Storage

File-Based Storage (JSON)

Users (`users.json`):

```
{
  "email@example.com": {
    "name": "User Name",
    "password": "scrypt:...",
    "firebase_uid": "...",
    "created_at": "2025-...",
    "auth_provider": "firebase"
```

```
  }
}
```

Payments (`payments/payments.json`):

```
{
  "email@example.com": [
    {
      "payment_id": "...",
      "order_id": "...",
      "amount": 19900,
      "status": "captured",
      "timestamp": "..."
    }
  ]
}
```

Assessment Attempts (`attempts/{attempt_id}.json`):

```
{
  "id": "...",
  "user": "email@example.com",
  "start": "2025-...",
  "questions": [...],
  "answers": {"1": "FAR", "2": "BM", ...},
  "submitted": true,
  "results": {
    "scores": {"FAR": 25, "BM": 30, "CRM": 20, "MO": 25},
    "strongest": "BM",
    "attributes": {...}
  }
}
```

Excel Files

Contact Submissions (`data/contacts/contacts.xlsx`): - Full Name, Email, Phone, Inquiry Type, Background, Message, Submitted At

Career Applications (`data/careers/careers.xlsx`): - Full Name, Email, Phone, Position, Trainings, Current CTC, etc. - Linked resume/cover letter files

Firebase Firestore

Collections: - `contacts` - Contact form submissions - `career_applications` - Career application submissions

Structure: - Documents include all form fields - Timestamp fields for submission tracking - Status fields for admin management

File Storage

Resumes/Cover Letters (`data/careers/`): - Format: `RESUME_{timestamp}_{original_filename}` - Format: `CL_{timestamp}_{original_filename}`

Deployment

Production Setup

1. **Use Gunicorn**:

   ```
   gunicorn app:app --bind 0.0.0.0:8000
   ```

2. **Environment Variables** (Recommended):

   - `FLASK_SECRET_KEY` - Session secret
   - `RAZORPAY_KEY_ID` - Payment gateway key
   - `RAZORPAY_KEY_SECRET` - Payment gateway secret
   - `FIREBASE_CRED_PATH` - Firebase credentials path

3. **Production Checklist**:

   ☐ Change `app.secret_key` to secure random value
   ☐ Move Razorpay keys to environment variables
   ☐ Set up proper logging
   ☐ Configure HTTPS
   ☐ Set up regular backups of JSON files
   ☐ Monitor Firebase usage/quota
   ☐ Change admin credentials

Procfile (Heroku/Similar)

```
web: gunicorn app:app
```

Configuration

Key Configuration Values

Flask App (lines 38-44):

```
app.secret_key = 'unicorn-secret-please-change'  # CHANGE THIS!
app.config['DEBUG'] = False
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
```

Razorpay (lines 236-239):

```
RAZORPAY_KEY_ID = "rzp_live_RRLzuRNwQiqFcR"
RAZORPAY_KEY_SECRET = "1Fct2RgdkxW97AWMTTRYsunC"
ASSESSMENT_PRICE = int(os.environ.get('ASSESSMENT_PRICE', '19900'))
```

Firebase (lines 57-68): - Credentials file: `firebase-service-account.json` - Auto-initializes if file exists

Assessment (line 24): - Question file: `Sample Assessment questions.xlsx`

Troubleshooting

Common Issues

1. **"Token used too early" Error**
   - **Cause**: Clock skew between client and server
   - **Solution**: Already handled with clock skew tolerance (60 seconds)
   - **Location**: `app.py` lines 109-166
2. **Receipt Length Error**
   - **Cause**: Razorpay receipt ID > 40 characters
   - **Solution**: Fixed with receipt ID shortening
   - **Location**: `app.py` lines 494-507
3. **Firebase Not Working**
   - **Check**: `firebase-service-account.json` exists
   - **Check**: File has valid credentials
   - **Location**: `app.py` lines 59-68
4. **Payment Verification Fails**
   - **Check**: Razorpay keys are correct
   - **Check**: Using correct environment (live/test)
   - **Location**: `app.py` lines 531-572
5. **Assessment Questions Not Loading**
   - **Check**: `Sample Assessment questions.xlsx` exists
   - **Check**: Excel file format is correct
   - **Location**: `app.py` lines 257-329

Debugging Tips

1. **Check Logs**: Look for print statements in code
2. **Session Issues**: Clear browser cookies
3. **File Permissions**: Ensure write access to data directories
4. **Firebase**: Check Firebase console for errors

Additional Notes

Security Considerations

1. **Change Secret Key**: Line 40 in `app.py`
2. **Admin Credentials**: Hardcoded, should be changed
3. **API Keys**: Consider using environment variables
4. **File Uploads**: Validate file types and sizes
5. **HTTPS**: Required for production

Future Enhancements

1. Database Migration: Consider moving from JSON to SQL database
2. Email Notifications: Add email sending for submissions
3. Admin Panel: Enhanced admin dashboard
4. Analytics: Add user analytics tracking
5. Testing: Add unit and integration tests

Contact Information

Path2Hire Contact: - Location: Kolkata - Email: contact@path2hire.com - Phone: +919051539665 - Website: www.path2hire.com

---

File Reference Quick Guide

File	Purpose	Key Functions
`app.py`	Main application	All routes, core logic
`career_report_generator.py`	Report generation	Scoring, PDF generation
`job_scraper.py`	Job scraping	Google Jobs scraping
`site/login.html`	Login UI	Firebase auth integration
`templates/assessment.html`	Assessment UI	Question display
`templates/results.html`	Results UI	Results display
`templates/payment.html`	Payment UI	Razorpay integration

Support & Maintenance

Key Areas to Monitor

1. **Payment Processing**: Monitor Razorpay dashboard
2. **Firebase Usage**: Monitor Firestore quota
3. **File Storage**: Monitor disk space for uploads
4. **Assessment Attempts**: Clean up old attempts periodically
5. **Error Logs**: Monitor application logs

Backup Strategy

- Backup `users.json` regularly
- Backup `payments/payments.json` regularly
- Backup assessment attempts (optional, can regenerate)
- Backup uploaded files in `data/careers/`

Document Version: 1.0
Last Updated: November 2025
Maintained By: Development Team