

ZipRecruiter Apply Webhook Documentation

ZR ATS Integration Team

Version 1.3, 2025-10

For the most up to date documentation, please download this PDF from our site.

Table of Contents

Introduction . . . 1
Requirements . . . 1
1. Webhook Specification . . . 2
1.1. Transport . . . 2
1.2. Format . . . 2
1.3. Response . . . 2
1.4. Fields . . . 3
1.5. JSON Schema . . . 5
1.6. Signatures . . . 5
1.6.1. Benefits . . . 5
1.6.2. Secret Exchange . . . 5
1.6.3. Full Signature Headers Example in HTTP Request . . . 6
1.6.4. Signature . . . 6
1.6.5. Caveat(s): . . . 9
2. Example Webhook Requests . . . 10
3. Frequently Asked Questions . . . 13
3.1. Do you provide a sandbox for testing? . . . 13
3.2. Do I need an authorization or API key? . . . 13
3.3. Can I redirect candidates to an external URL? . . . 13
3.4. Do I need a URL for every job? . . . 13
3.5. Can I add an interview or screening questions to my jobs? . . . 13
3.6. This webhook doesn’t meet my ATS requirements, can you make adjustments? . . . 13
Contact Information . . . 14
Appendix A: Structured Form Data . . . 15

Introduction

ZipRecruiter aims to make the job posting process simple from end to end. This includes automated job posting from your website or Applicant Tracking System (ATS) platform, as well as delivering new job applications back to it.

This provides a convenient way to post jobs on ZipRecruiter while being able to manage your candidate workflow from your preferred solution.

Requirements

ZipRecruiter Apply Webhook requires the following:

1. You can set up an integration using the Partner Job API or use an XML file for automatic job posting, in at least the minimal format. This allows us to post and unpost jobs in line with your calls, as well as provides us with your platform's job identifiers, so applicants can be delivered back to a specific job in your platform.

2. You must provide an HTTPS endpoint which accepts a POST request in JSON format. This is a single URL which handles submissions for all of your jobs. You will need to let us know which content type you need when you furnish the URL for this endpoint.

Setup for ZipRecruiter Apply Webhook will need to be coordinated through your account representative at this time, or you can contact us for help.

1. Webhook Specification

1.1. Transport

The ZipRecruiter Apply Webhook makes an HTTPS POST to the endpoint of your choosing. We do not deliver to non-HTTPS services.

This endpoint **must** furnish a verifiable SSL certificate signed by a well-known Certificate Authority (such as those trusted by Mozilla). If your web browser can visit the endpoint without generating a security warning, we can support it.

1.2. Format

The POST request body must be delivered as **JSON**, as requested during setup.

The `Content-Type` request header will be set accordingly to `application/json`.



Because JSON and form-data support stuctured data in different ways, there are slight differences in the data sent in each format. For the most part, they are simple key/value pairs, but see the [resume] and [answers] fields for more details.

1.3. Response

A `200 OK` or 2xx HTTP response is considered a successful delivery, while the response body is ignored. Other response codes are considered unsuccessful, and are logged and handled, and may be retried automatically or manually.

If we receive a 5xx response, the system will automatically retry delivery up to six times over a two-day period before marking the attempt as failed.



In order to ensure reliable delivery, your endpoint **must** return meaningful HTTP response codes to the webhook's request — a response that says `200 OK` but has an error message in the body is considered successful.

1.4. Fields

All field values are alphanumeric strings unless specified otherwise.

response_id

A unique identifier for the Job Application being transmitted, assigned by ZipRecruiter.

job_id

The external identification for the Job as provided to ZipRecruiter in a feed's referencenumber field, or other custom integration. This is the Job ID **your** ATS uses.

email

The Applicant's email address.

name

The Applicant's full name as entered.

first_name

The Applicant's first name as entered or parsed.

last_name

The Applicant's last name as entered or parsed.

phone

The Applicant's phone number. Optional.

But when it's part of the response, it will be of the format: +N NNNNNNNNN

resume

The Applicant's attached resume, in PDF format.

In JSON format, a base-64 encoded stream without whitespace.

In form-data format, a file upload attachment as described in RFC2388, with generated unique filename added in the header.

attributes

Key/value pairs for custom behavior. See also the tag `<partner_attributes>` in ZipRecruiter Feed Import. Optional.

A JSON object, with custom string fields and string values (i.e., no nesting).

answers

The Applicant's interview answers, if an interview was added to the job through your job feed. This is an advanced and optional, feature. If not in use, this field is omitted.

In all formats, `answers` is an array of objects.



For more information on attaching interviews and receving answers, please see our JSON Interview Documentation, and the Answer Specification therein.

profile

The jobseeker's profile data. The data is either parsed from the jobseeker's resume or entered fron profile page. If there is no profile data, profile field is omitted. The fields are either string or array of objects.

String fields: mobile, executive_summary, objective, text_resume

Array of objects: job_records, education_records, achievement_records, license_certification_records, association_records

post_url

Optional. A feed node to be used if the customer wants to provide a dynamic candidate submission endpoint.

zr_application_id

A unique internal identifier to ZipRecruiter for the Job Application being transmitted.

1.5. JSON Schema

You can view the schema for our JSON Screener Question Specification v2 below, which leverages our API Webhook delivery.

JSON Schema v2:

The new version of this schema can be found at the URL below.

- https://static.ziprecruiter.com/schema/apply-webhook-schema-v2.json

The following tools can be used from a web browser to test a JSON document against a JSON Schema.

- JSON Schema Lint
- JSON Schema Validator

Be aware that JSON Schema is currently in an ongoing draft process, and tools may not support the most recent draft specification. Most schemas can be translated back to older drafts in order to support outdated tools.

1.6. Signatures

You may optionally enable signatures for network requests from Apply Webhook.

After signatures are enabled, you must:

1. Verify the signature.
2. Check that the current time is within a few seconds of the timestamp in the request.

1.6.1. Benefits

- Guarantee that payloads sent from Apply Webhook originate from ZipRecruiter.
- Guarantee that payloads sent from Apply Webhook were not changed.
- Prevent replay attacks.

1.6.2. Secret Exchange

Upon request, we will provide a secret string via a one-time access link to you, leveraging a secure delivery method. This secret will be shared within the same ZipRecruiter org. You should keep this secret safe from potentially malicious actors.

1.6.3. Full Signature Headers Example in HTTP Request

POST http://localhost/apply
Accept-Encoding: gzip
Authorization: Basic 123abc
User-Agent: Test-WWW-Mechanize/1.50
Content-Length: 22978
Content-Type: application/json
X-ZipRecruiter-Signature: c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk=
X-ZipRecruiter-Signature-Timestamp: 2021-02-12T15:49:55.214Z
X-ZipRecruiter-Signature-Version: v1; sha256

{"response_id" : "1","first_name" : "John", "last_name" : "Doe"}

1.6.4. Signature

Once signatures are enabled and secrets have been exchanged, we will send 3 signature-related headers with every Apply Webhook delivery to the your endpoint: "X-ZipRecruiter-Signature", "X-ZipRecruiter-Signature-Version", and "X-ZipRecruiter-Signature-Timestamp".

X-ZipRecruiter-Signature: <signature>

<signature>: String of characters returned by signature algorithm.

Example: c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk=

You can verify this signature as follows:

1. Take the Apply Webhook payload. For example:
    - {"response_id" : "1","first_name" : "John", "last_name" : "Doe"}
2. Take the timestamp from X-ZipRecruiter-Signature-Timestamp:
    - 2021-02-12T15:49:55.214Z"
3. Concatenate timestamp and payload, separated by a dot (.). Note that the payload needs to be from the request body without altering it.
    - 2021-02-12T15:49:55.214Z.{"response_id" : "1","first_name" : "John", "last_name" : "Doe"}
4. Create a base64-encoded HMAC SHA256 hash out of timestamp.payload with your secret without breaking the encoded string into lines.
    - c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk=
5. Verify that the calculated signature is identical to the signature in X-ZipRecruiter-Signature with a case-sensitive equality check.

Java Example - Checking Signature

```
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
public class Main {
    public static String encode(String key, String data) {
        Mac sha256_HMAC = null;
        try {
            sha256_HMAC = Mac.getInstance("HmacSHA256");
        }
        catch(NoSuchAlgorithmException e) {
            System.out.println("Something is wrong");
        }
        SecretKeySpec key_spec = new SecretKeySpec(key.getBytes(StandardCharsets
.US_ASCII), "HmacSHA256");
        try {
            sha256_HMAC.init(key_spec);
        }
        catch(InvalidKeyException e) {
            System.out.println("Something is wrong");
        }
        return Base64.getEncoder().encodeToString(sha256_HMAC.doFinal(data.getBytes
(StandardCharsets.US_ASCII)));
    }
    public static void main(String[] args) {
        String message = getMessage();
        String secret = "my secret";
        String signature = encode(secret,message);
        String sent_signature = "c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk=";
        System.out.println(sent_signature.equals(signature));
    }
}
```

Python Example - Checking Signature

```
import hmac
import hashlib
import base64
message = getMessage()
secret = 'my secret'
signature = base64.b64encode(hmac.new(bytes(secret , 'ascii'), msg = bytes(message ,
'ascii'), digestmod = hashlib.sha256).digest()).decode()
sent_signature = 'c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk=';
print(signature == sent_signature)
```

Go Example - Checking Signature

```
import (
    b64 "encoding/base64"
    "fmt"
    "crypto/sha256"
    "crypto/hmac"
)
func main() {
    message := getMessage()
    secret := "my secret"
    mac := hmac.New(sha256.New, []byte(secret))
    mac.Write([]byte(message))
    signature := b64.StdEncoding.EncodeToString(mac.Sum(nil))
    sent_signature := "c//+/Qo4wuJyzFENikeVOabKFXCJ5wLHzlTqkxHgZTk="
    fmt.Println(signature == sent_signature)
}
```

X-ZipRecruiter-Signature-Version: <version; comment>

<version; comment>: a version followed by an ignorable comment, separated by a semi-colon (;). The comment indicates the hashing algorithm. Versioning indicates our method for computing signatures.

Example: *v1; sha256*

X-ZipRecruiter-Signature-Timestamp: <timestamp>

<timestamp>: a date-timestamp in UTC conforming to RFC 3339.

Example: *2021-02-12T15:49:55.214Z*

To prevent replay attacks, you should check that the timestamp in the header is within a few seconds of the current timestamp. Too large of a window size will result in false negatives (i.e. not catching a replay attack if one has happened). Too small of a window size will result in false positives (i.e. rejecting valid requests).

Java Example - Checking Timestamp

```java
import java.util.Calendar;
import java.time.Instant;
import java.time.Duration;
public class Main {
    public static void main(String[] args) {
        Instant sent_time = get_time();
        Instant current_time = Calendar.getInstance().toInstant();
        Duration duration = Duration.ofSeconds(5);
        Instant max_window = sent_time.plus( duration );
        System.out.println(current_time.isBefore(max_window));
    }
}
```

Python Example - Checking Timestamp

```python
from datetime import datetime, timedelta
sent_time = get_time()
current_time = datetime.now()
max_window = sent_time + timedelta(0,5)
print(current_time < max_window)
```

Go Example - Checking Timestamp

```go
import (
    "fmt"
    "time"
)
func main() {
    sent_time := get_time()
    current_time :=  time.Now()
    max_window := sent_time.Add(time.Second * time.Duration(5))
    fmt.Println(current_time.Before(max_window))
}
```

1.6.5. Caveat(s):

- Signatures are only supported for JSON data

2. Example Webhook Requests

The following listings use an example endpoint address of `https://yourhost.example.com/job/apply`, and include extra whitespace or other insignificant omissions for readability here.

Example JSON Request

```
POST /job/apply HTTP/1.1
Host: yourhost.example.com
Content-Type: application/json

{
"response_id": "a39bd9a",
"zr_application_id": "2770c22a-ab43-43a1-b541-f9650727b0df",
"job_id": "1000002”,
"name": "Tom Foolery",
"first_name": "Tom",
"last_name": "Foolery",
"email": "tf@example.org",
"phone": "+1 5555551942",
"resume":
"JVBERi0xLjUKJb/3ov4KMiAwIG9iago8PCAvTGluZWFyaXplZCAxIC9MIDE3ODA3IC9IIFsgNjg3IDEyNiBdI
C9PIDYgL0UgMTc1MzIgL04gMSAvVCA...jIxNgfxdslJUVPRgo=",
"attributes": {
        "post_url": "https://backend.example.com/api/ziprecruiter/12345678?
recruiterEmail=tf@example.org",
         "tracking_code": "r2d2c3po"
},
"profile": {
       "executive_summary": "Business Analyst with over 5 years of experience supporting
                             business solution software and analyzing business operations.",
         "mobile": "+1 555 555-1942",
         "job_records": [
             {
                 "start_date_precision": "YearMonthDay",
                 "position": "Business Analyst",
                 "description": "Analyzing data",
                 "start_date": "2012-09-01",
                 "end_date_precision": null,
                 "employer": "Toms",
                 "current": 1,
                 "end_date": null
             },
             {
                 "position": "Business Analyst",
                 "start_date_precision": "YearMonthDay",
                 "start_date": "2010-09-01",
                 "end_date_precision": "YearMonthDay",
                 "description": "Still Analyzing data",
                 "current": "0",
                 "employer": "Fooleries",
                 "end_date": "2012-05-01"

             }
         ]}
}
```

3. Frequently Asked Questions

3.1. Do you provide a sandbox for testing?

Unfortunately, at the moment we do not have a sandbox for testing the integration. However, we can make arrangements to provide a test import and a few test applications if needed.

3.2. Do I need an authorization or API key?

No. If the endpoint you provide to us requires special HTTP headers or HTTP Basic Authencation, though, we can support that.

3.3. Can I redirect candidates to an external URL?

No. Our integrations require that candidates use the ZipRecruiter Apply process.

3.4. Do I need a URL for every job?

No. Please provide only one POST URL for the webhook, and the software receiving that POST can associate applications with jobs by the provided [job_id] field.

3.5. Can I add an interview or screening questions to my jobs?

This is an advanced feature. We support a newly updated question format, as well as delivery of applicant answers through this webhook. If you are interested, please contact us. If your ATS platform supports interviews but doesn't support our format, please reach out and we may be able to help on a case-by-case basis. Also, please reach out if you intend to use the new question format.

3.6. This webhook doesn't meet my ATS requirements, can you make adjustments?

Please contact us about this, as we may be able to support anything from minor tweaks to major custom integrations.

Contact Information

If you have questions about the ZipRecruiter Apply Webhook integration, please don't hesitate to contact us.

Email: atsintegrations@ziprecruiter.com

If you have questions about ATS Integrations and Partnerships, contact us or find more info here:

Web: https://www.ziprecruiter.com/ats-partners
Email: atsintegrations@ziprecruiter.com

Appendix A: Structured Form Data



Support for structured form data is only necessary if you are using the form-data format **and** are expecting [answers] fields for responses to attached job interviews.

In order to map structured data that can be sent as-is in JSON to something suitable for form-data transmission, we use a subscript mapping style as popularized by Ruby's Rails and Rack projects.

That is, while reading form keys, an empty subscript `[]` indicates the key is part of an array, a subscript with text `[id]` indicates the key is a key nested inside an object, and encountering a nested key that already exists indicates a new object should be created.

Following this mapping, this JSON snippet and list of key/value pairs are equivalent and can be translated back and forth:

JSON

```
{
  "items": [
    {"id": "4", "status": "available"},
    {"id": "17", "status": "backordered"},
  ]
}
```

Pairs

```
'items[][id]'     ⇒ '4',
'items[][status]' ⇒ 'available',
'items[][id]'     ⇒ '17',
'items[][status]' ⇒ 'backordered'
```

Answers will be structured similarly to the above, see Example Answers for more information. In the future, more structured fields may be added, but the preexisting core fields will not be changed.