Offers Documentation

Top Free Games

May 16, 2018

Contents

Contents:

CHAPTER 1

Overview

Offers makes the implementation and management of offers in our games an predictable and scalable task.

Used to sell special packs of the game's items. Offers helps doing A/B testing, remote configurations of offers, scheduling and monitoring.

1.1 Features

- **Create Offers** - Create a template with information about a game's offer: what it gives to the player, when it is enabled, how many times a player can see and buy it;
- **Get Available Offers** - Given a player and a game, returns the available offers for each placement in the UI;
- **New Relic Support** - Natively support new relic with segments in each API route for easy detection of bottlenecks;

1.2 Architecture

Offers is composed of an API responsible for creation of games and templates, and retrieval of offers for each player.

1.3 The Stack

Our code is in Golang, with:

- Database - Postgres >= 9.5;

1.4 Who's Using it

Well, right now, only us at TFG Co, are using it, but it would be great to get a community around the project. Hope to hear from you guys soon!

1.5 How To Contribute?

Just the usual: Fork, Hack, Pull Request. Rinse and Repeat. Also don't forget to include tests and docs (we are very fond of both).

CHAPTER 2

Hosting Offers

2.1 Docker

Running Offers with docker is rather simple. Our docker container image comes bundled with the API binary. All you need to do is load balance all the containers and you're good to go. The API runs at port `8888` in the docker image.

Offers uses PostgreSQL to store offers information. The container takes environment variables to specify this connection:

- `OFFERS_POSTGRES_HOST` - PostgreSQL host to connect to;
- `OFFERS_POSTGRES_PORT` - PostgreSQL port to connect to;
- `OFFERS_POSTGRES_DBNAME` - PostgreSQL database to connect to;
- `OFFERS_POSTGRES_PASSWORD` - Password of the PostgreSQL Server to connect to;
- `OFFERS_POSTGRES_USER` - PostgreSQL user;

Offers uses basic auth to restrict access to routes that are not used directly by a client consuming the offers.

- `OFFERS_BASICAUTH_USERNAME` - Basic Auth user;
- `OFFERS_BASICAUTH_PASSWORD` - Basic Auth password;

When a client requests the available offers the API returns a `max-age` header. The cache TTL (in seconds) can be defined using the following variable:

- `OFFERS_CACHE_MAXAGESECONDS` - Max age in seconds;

Other than that, there are a couple more configurations you can pass using environment variables:

- `OFFERS_NEWRELIC_KEY` - If you have a New Relic account, you can use this variable to specify your API Key to populate data with New Relic API;
- `OFFERS_NEWRELIC_APP` - Name of the NewRelic app ;
- `OFFERS_SENTRY_URL` - If you have a sentry server you can use this variable to specify your project's URL to send errors to.

CHAPTER 3

Offers API

3.1 Healthcheck Routes

3.1.1 Healthcheck

`GET /healthcheck`

Validates if the app is still up, including its database connection.

- Success Response
  - Code: `200`
  - Content:

```
{
  "healthy": true
}
```

- Error Response It will return an internal error if it failed to connect to the database.
  - Code: `500`
  - Content:

```
{
  "healthy": false
  "error": "DatabaseError",
  "code": "OFF-000",
  "description": [string]    // error description
}
```

3.2 Game Routes

3.2.1 Upsert Game

`PUT /games/:id`

Updates an existing Game or insert a new Game into database. `:id` must match `^[^-][a-zA-Z0-9-_]*$`.

Requires basic auth.

- Payload

```
{
  "name":     [string], // required, 255 characters max
  "metadata": [json]    // optional
}
```

  - Field Descriptions
    - **id**: Unique ID that identifies the game
    - **name**: Prettier game identifier to show on UI
    - **metadata**: Any additional information one would like to access later

- Metadata

  The metadata field is optional, but there are two keys that have direct impact in the `GET /available-offers` route.

```
{
  cacheMaxAge: <ttl in seconds>,
  allowInefficientQueries: <bool>
}
```

  - Field Descriptions
    - **cacheMaxAge**: TTL in seconds returned in the `Cache-Control max-age` header. If not configured in the game, offers-api default value will be used.
    - **allowInefficientQueries**: If set to true the API will match offers containing filters with intervals (`gte`, `lt`) and offers without any filters. This is less efficient because these queries do not make proper use of GIN index.

- Success Response
  - Code: `200`
  - Content:

```
{
  "gameId": [string]
}
```

- Error response
  - If missing or invalid arguments
    - Code: `422`
    - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

  - It will return an error if the query on db (upsert) failed
    * Code: `500`
    * Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.2.2 List Games

`GET /games`

Lists all existing games.

Requires basic auth.

- Success Response
  - Code: `200`
  - Content:

```
[
  {
    "id":       [string],
    "name":     [string],
    "metadata": [json]
  },
  ...
]
```

- Error response

  It will return an error if the query on db failed
  - Code: `500`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.3 Offer Routes

3.3.1 Create Offer

`POST /offers`

Inserts a new Offer into the database.

Requires basic auth.

- Payload

```
{
  "name":      [string], // required, 255 characters max
  "productId": [string], // 255 characters max, required if cost is not defined
  "cost":      [json],   // required if productId is not defined
  "gameId":    [string], // required, matches ^[^-][a-zA-Z0-9-_]*$
  "contents":  [json],   // required
  "placement": [string], // required, 255 characters max
  "period":    {         // required
    "every": [string],   // required
    "max":   [int]       // required
  },
  "frequency": {         // required
    "every": [string],   // required
    "max":   [int]       // required
  },
  "trigger":   {         // required
    "from":  [int],      // required
    "to":    [int]       // required
  },
  "metadata":  [json],   // optional
  "filters":   [json]    // optional
}
```

  - Field Descriptions

    * **name**: Prettier game identifier to show on UI.
    * **productId**: Identifier of the item to be bought on PlayStore or AppStore. It is required if cost is not set.
    * **cost**: A JSON indicating the offer cost in terms of the game currency (ex.: { “gems”: 500 }). It is required if productId is not set.
    * **gameId**: ID of the game this template was made for (must exist on Games table on DB).
    * **contents**: What the offer provides (ex.: { “gem”: 5, “gold”: 100 }).
    * **metadata**: Any information the Front wants to access later.
    * **period**: Enable player to buy offer every x times, at most y times. every: decimal number with unit suffix, such as “300ms”, “-1.5h” or “2h45m”. Valid time units are “ns”, “us” (or “µs”), “ms”, “s”, “m”, “h”max: maximum number of times this offer can be bought by the playerIf “every” is an empty string, then the offer can be bought max times with no time restriction. If “max” is 0, then the offer can be bought infinite times with time restriction. They can’t be “” and 0 at the same time.
    * **frequency**: Enable player to see offer on UI x/unit of time, at most y times. every: decimal number with unit suffix, such as “300ms”, “-1.5h” or “2h45m”. Valid time units are “ns”, “us” (or “µs”), “ms”, “s”, “m”, “h”max: maximum number of times this offer can be seen by the playerIf “every” is

an empty string, then the offer can be seen max times with no time restriction. If "max" is 0, then the offer can be seen infinite times with time restriction. They can't be "" and 0 at the same time.

* **trigger**: Time when the offer is available.
* **filters**: The filters for the offer, they can be of three different types for a given attribute: interval: the attribute must define the beginning and/or end of the interval with "geq" and "lt", the interval includes the beginning but not the endequality: the attribute must define the "eq", the value that the filter expects the attribute to be equal to, it should be a stringdifference: the attribute must define "neq", the value that the filter expects the attribute to be different from, it should be a stringAn example: "{ "intervalValue": { "geq": 0.0, "lt": 10.0 }, "equalValue": { "eq": "John" } }". Please note that interval filters are only enabled if the game has the `allowInefficientQueries` property set to `true`.
* **enabled**: True if the offer is enabled.
* **placement**: Where the offer is shown in the UI.
* **version**: Offer current version.

- Success Response
  - Code: `200`
  - Content:

```
{
  "id":         [uuidv4],   // offer unique identifier
  "name":       [string],
  "productId": [string],
  "cost":       [json],
  "gameId":     [string],
  "contents":   [json],
  "metadata":   [json],
  "placement": [string],
  "period":     {
    "every": [string],
    "max":    [int]
  },
  "frequency": {
    "every": [string],
    "max":    [int]
  },
  "trigger":    {
    "from":   [int],
    "to":     [int]
  },
  "enabled":    true,       // created offer are enabled by default
  "version":    1,          // created offer version is 1 by default
  "filters":    [json]
}
```

- Error response

  It will return an error if the request has missing or invalid arguments

  - Code: `422`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

It will return an error if the query on db (insert) failed

- Code: `500`
- Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.3.2 Update Offer

`PUT /offers/:id`

Updates the offer with given id in the database.

Requires basic auth.

- Payload

```
{
  "name":      [string], // required, 255 characters max
  "productId": [string], // 255 characters max, required if cost is not defined
  "cost":      [json],   // required is productId is not defined
  "gameId":    [string], // required, matches ^[^-][a-zA-Z0-9-_]*$
  "contents":  [json],   // required
  "placement": [string], // required, 255 characters max
  "period":    {         // required
    "every": [string],   // required
    "max":   [int]       // required
  },
  "frequency": {         // required
    "every": [string],   // required
    "max":   [int]       // required
  },
  "trigger":   {         // required
    "from":  [int],      // required
    "to":    [int]       // required
  },
  "metadata":  [json],   // optional
  "filters":   [json]    // optional
}
```

  - Field Descriptions
    - **name**: Prettier game identifier to show on UI.
    - **productId**: Identifier of the item to be bought on PlayStore or AppStore. It is required if cost is not set.

    * **cost**: A JSON indicating the offer cost in terms of the game currency (ex.: { “gems”: 500 }). It is required if productId is not set.
    * **gameId**: ID of the game this template was made for (must exist on Games table on DB).
    * **contents**: What the offer provides (ex.: { “gem”: 5, “gold”: 100 }).
    * **metadata**: Any information the Front wants to access later.
    * **period**: Enable player to buy offer every x times, at most y times. every: decimal number with unit suffix, such as “300ms”, “-1.5h” or “2h45m”. Valid time units are “ns”, “us” (or “µs”), “ms”, “s”, “m”, “h”max: maximum number of times this offer can be bought by the playerIf “every” is an empty string, then the offer can be bought max times with no time restriction. If “max” is 0, then the offer can be bought infinite times with time restriction. They can’t be “” and 0 at the same time.
    * **frequency**: Enable player to see offer on UI x/unit of time, at most y times. every: decimal number with unit suffix, such as “300ms”, “-1.5h” or “2h45m”. Valid time units are “ns”, “us” (or “µs”), “ms”, “s”, “m”, “h”max: maximum number of times this offer can be seen by the playerIf “every” is an empty string, then the offer can be seen max times with no time restriction. If “max” is 0, then the offer can be seen infinite times with time restriction. They can’t be “” and 0 at the same time.
    * **trigger**: Time when the offer is available.
    * **filters**: The filters for the offer, they can be of three different types for a given attribute: interval: the attribute must define the beginning and/or end of the interval with “geq” and “lt”, the interval includes the beginning but not the endequality: the attribute must define the “eq”, the value that the filter expects the attribute to be equal to, it should be a stringdifference: the attribute must define “neq”, the value that the filter expects the attribute to be different from, it should be a stringAn example: “{ “intervalValue”: { “geq”: 0.0, “lt”: 10.0 }, “equalValue”: { “eq”: “John” } }”.
    * **enabled**: True if the offer is enabled.
    * **placement**: Where the offer is shown in the UI.
    * **version**: Offer current version.

- Success Response
  - Code: `200`
  - Content:

```
{
  "id":      [uuidv4],   // offer unique identifier
  "version": [int]       // updated offer version
}
```

- Error response

  It will return an error if the offer with given id does not exist in the database

  - Code: `404`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

  It will return an error if the request has missing or invalid arguments

  - Code: `422`

- Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

It will return an error if the query on db (insert) failed

- Code: `500`
- Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.3.3 Enable offer

`PUT /offers/:id/enable?game-id=<required-game-id>`

Enables an offer. `:id` must be an `uuidv4`.

Requires basic auth.

- Success Response
  - Code: `200`
  - Content:

```
{}
```

- Error Response

  It will return status code 404 if the offer with given ID does not exist

  - Code: `404`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

  It will return status code 500 internal error occurred

  - Code: `500`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.3.4 Disable offer template

`PUT /offers/:id/disable?game-id=<required-game-id>`

Disables an offer template. `:id` must be an `uuidv4`.

Requires basic auth.

- Success Response
  - Code: `200`
  - Content:

```
{}
```

- Error Response

  It will return status code 404 if the offer with given ID does not exist

  - Code: `404`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

  It will return status code 500 internal error occurred

  - Code: `500`
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.3.5 List Offers

`GET /offers?game-id=<required-game-id>&limit=<optional-limit>&offset=<optional-offset>`

- game-id: the given game id.
- limit: how many offers will be returned (the page size); default is 50.
- offset: the page number; default is 0.

Lists all game's offers.

Requires basic auth.

- Success Response
  - Code: `200`
  - Content:

```
{
  "offers": [
    {
      "id":         [uuidv4],   // offer template unique identifier
      "key":        [uuidv4],
      "name":       [string],
      "productId": [string],
      "cost":       [json],
      "gameId":     [string],
      "contents":   [json],
      "metadata":   [json],
      "placement": [string],
      "period":     {
        "every": [string],
        "max":   [int]
      },
      "frequency": {
        "every": [string],
        "max":   [int]
      },
      "trigger":    {
        "from":   [int],
        "to":     [int]
      },
      "enabled":    [bool],
      "version":    [int],
      "filters":    [json]
    },
    ...
  ],
  "pages": [int]
}
```

- Error response

  It will return an error if the query on db failed

  - Code: `500`
  - Content:

    ```
    {
      "error": [string],       // error
      "code":  [string],       // error code
      "description": [string]  // error description
    }
    ```

3.4 Offer Request Routes

There are the routes accessed by the offers lib.

3.4.1 Get Available Offers

```
GET /available-offers?player-id=<required-player-id>&game-id=<required-game-id>&<attr1>=<va
..
```

Gets the available offers for a player of a game. An offer is available if it respects the frequency (last time player saw the offer), respects the period (last time player claimed the offer), is triggered (current time is between "from" and "to"), matches the filters of the offer for the parameters sent in the query string and is enabled. The success response is a JSON where each key is a placement on the UI and the value is a list of available offers.If an attribute sent in the query string doesn't exist in a filter it is ignored and the extra parameters for a filter are ignored if the request doesn't send a value for them. If the filter defines an interval the query string parameter value must be a number. There is no limit in the amount of attributes that can be sent to be used in the filters.

- Success Response
  - Code: `200`
  - Content:

```
  {
    "placement-1": [
      {
          "id":                    [uuidv4], // offer id
          "productId":             [string], // 255 characters max
          "cost":                  [json],   // offer cost as registered in␣
↪the offer template
          "contents":              [json],   // offer contents as registered␣
↪in the offer template
          "metadata":              [json],   // offer metadata as registered␣
↪in the offer template
          "expireAt":              [int64]   // timestamp (seconds since␣
↪epoch) until when the offer is valid
      },
      ...
    ]
    "placement-2": [
      ...
    ],
    ...
  }
```

  - Header: A max-age header is sent to indicate how long the response returned by get available offers can be cached.

```
Cache-Control: max-age=<seconds>
```

- Error Response
  - Code: `400`, if player-id is not informed
  - Code: `400`, if game-id is not informed
  - Code: `500`, if server failed in any other way
  - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.4.2 Claim Offer

`PUT /offers/claim`

Claims a player's offer. Should only be called after payment confirmation. `:id` must be an `uuidv4`.

- Payload

```
  {
    "gameId":   [string],      // required, matches ^[^-][a-zA-Z0-9-_]*$
    "playerId": [string],      // required, 255 characters max
    "productId": [string],     // 255 characters max, required if id is not␣
↪defined
    "timestamp": [int64],      // required, unix timestamp of the purchase
    "transactionId": [string], // required, unique identifier of the purchase
    "id": [uuidv4]             // optional, the id of the offer being claimed,␣
↪required if productId is not defined
  }
```

  If the id of the offer being claimed is sent it will be used to find the offer, increment the claim counter and the timestamp of the last time this offer was claimed. If not, the other information present in the payload will be used to try to identify the offer that is being claimed.

- Success Response
  - Code: `200`
  - Content: if the player can still see the offer:

```
  {
    "contents": [json]
    "nextAt": [int64]  // unix timestamp of the next time the offer can be␣
↪shown
  }
```

    if the player can no longer see the offer

```
  {
    "contents": [json]
  }
```

- If the player claimed an offer that he already claimed its contents are returned but with another status code, so the caller can decide whether to give the player the offer contents or not.
  - Code: `409`
  - Content: if the player can still see the offer:

```
  {
    "contents": [json]
    "nextAt": [int64]  // unix timestamp of the next time the offer can be␣
↪shown
  }
```

    if the player can no longer see the offer

```
  {
    "contents": [json]
  }
```

- Error Response

  - If a offer with id, gameId and playerId was not found in database.
    * Code: `404`
    * Content:

```
{
  "error": [string],        // error
  "code":  [string],        // error code
  "description": [string]   // error description
}
```

  - If any internal error occurred.
    * Code: `500`
    * Content:

```
  {
    "error": [string],        // error
    "code":  [string],        // error code
    "description": [string]   // error description
  }
```

3.4.3 Offer Impressions

`PUT /offers/:id/impressions`

Updates the time when the offer was last seen by the player and increments an impressions counter. `:id` must be an `uuidv4`.

- Payload

```
{
  "gameId":   [string],   // required, matches ^[^-][a-zA-Z0-9-_]*$
  "playerId": [string],   // required, 255 characters max
  "impressionId" [uuidv4] // required, unique identifier for this impression
}
```

  The `impressionId` field is used so this request can be idempotent. If more than one request is sent with the same `impressionId` the counter and the last impression timestamp will not be updated.

- Success Response
  - Code: `200`
  - Content: if the player can still see the offer:

```
  {
    "nextAt": [int64]  // unix timestamp of the next time the offer can be␣
↪shown
  }
```

    if the player can no longer see the offer

```
  {}
```

- Conflict Response (if the `impressionId` was already sent in a previous request):
  - Code: `200`

  - Content: if the player can still see the offer:

```
  {
    "nextAt": [int64]  // unix timestamp of the next time the offer can be␣
↪shown
  }
```

    if the player can no longer see the offer:

```
  {}
```

- Error Response
  - If missing or invalid arguments.
    - Code: `422`
    - Content:

```
  {
    "error": [string],       // error
    "code":  [string],       // error code
    "description": [string]  // error description
  }
```

  - If a offer with id, gameId and playerId was not found in database.
    - Code: `404`
    - Content:

```
  {
    "error": [string],       // error
    "code":  [string],       // error code
    "description": [string]  // error description
  }
```

  - If any internal error occurred.
    - Code: `500`
    - Content:

```
{
  "error": [string],       // error
  "code":  [string],       // error code
  "description": [string]  // error description
}
```

3.4.4 Get Offer Info

`GET /offer-info?player-id=<required-player-id>&game-id=<required-game-id>&offer-id=<require`

Gets information about a specific offer. This route can be used by a player that for some reason lost the information of a still valid offer and wants to retrieve it. The success response is a JSON object with the offer's attributes.

- Success Response
  - Code: `200`
  - Content:

```
  {
      "id":                    [uuidv4], // offer id
      "productId":             [string], // 255 characters max
      "cost":                  [json],   // offer cost as registered in the␣
↪offer template
      "contents":              [json],   // offer contents as registered in␣
↪the offer template
      "metadata":              [json],   // offer metadata as registered in␣
↪the offer template
      "expireAt":              [int64]   // timestamp (seconds since epoch)␣
↪until when the offer is valid
  }
```

  - Header: A max-age header is sent to indicate how long the response returned by get offer info can be cached.

```
Cache-Control: max-age=<seconds>
```

- Error Response
  - Code: `400`, if player-id is not informed
  - Code: `400`, if game-id is not informed
  - Code: `400`, if offer-id is not informed

* Code: `404`, if the offer was not found

* Code: `500`, if server failed in any other way * Content: `{ "error": [string], // error "code": [string], // error code "description": [string] // error description }`

CHAPTER 4

Indices and tables

- genindex
- modindex
- search