Boghche Documentation

Release

Sepehr Hamzehlouy Nime Ajdari Mehdi Makhdoumi

May 04, 2018

Contents

Boghche is another but complete cms. Its server is written in python and has Vue + Nuxt, iOS and Android clients. Currently it has a complete blog and a complete eCommerce. Its goal is to make costumization very easy and simple.

CHAPTER 1

Installation

1.1 Server Installation

WIP

1.2 Web Installation

WIP

To use Boghche server you can use Restful APIs documentation listed below.

CHAPTER 2

Restful APIs

2.1 Authentication

2.1.1 Objects

Object are json objects.

1. **Permission**

   name Name of the permission to display it.

   code_name the code for the permission.

2. **Group**

   id ID of the group in database.

   name Name of the group to display it.

   permissions List of permission object.

3. **User**

   id ID of the user in database.

   first_name First name of the user.

   last_name Last name of the user.

   email email of the user.

4. **Access** At login API, if the user is staffmember or admin we have such an object.

   group The group object which user is member of it.

   is_admin If the user is admin then its true.

   is_staff If the user is staffmember then its true.

2.1.2 Errors

If a token needs to be refreshed:

status_code 401

response

```
{"message": "Fresh token required"}
```

Invalid token:

status_code 422

response

```
{"message": "reason"}
```

Expired token:

status_code 401

response

```
{"message": "Token has expired"}
```

Revoked token:

status_code 401

response

```
{"message":"Token has been revoked"}
```

All of the listed APIs are restfull and content-type is `application/json`

2.1.3 Registration API

class fardel.core.auth.views.**RegistrationApi**

URL /api/auth/register/

post()

Required arguments

- email
- password

Optional arguments

- first_name
- last_name

Response

```
{
   "message":"Successfully registered",
   "access_token":"access_token",
   "refresh_token":"refresh_token"
}
```

Errors if email or password does not provided:

status_code 400

response

```
{"message":"Unvalid form submitted"}
```

If email already exists:

status_code 409

response

```
{"message": "A user with this email already exists."}
```

2.1.4 Login API

class fardel.core.auth.views.**LoginApi**

URL /api/auth/login/

post()

Required arguments

- email
- password

Response

```
{
   "message":"Successfully registered",
   "access_token":"access_token",
   "refresh_token":"refresh_token",
   "access": AccessObject
}
```

Errors if email or password does not provided:

status_code 400

response

```
{"message":"Unvalid form submitted"}
```

If email or password is not correct:

status_code 401

response

```
{"message":"Username or password is not correct"}
```

2.1.5 Logout API

class fardel.core.auth.views.**LogoutApi**

URL /api/auth/logout/

post()

- Authorization header containing access token is required

Status_code 200

Response

```
{
    "message": "Access token has been revoked"
}
```

2.1.6 Logout Refresh Token API

class fardel.core.auth.views.**LogoutRefreshApi**

URL /api/auth/logout-refresh/

post()

- Authorization header containing refresh token is required

Status_code 200

Response

```
{
    "message": "Refresh token has been revoked"
}
```

2.1.7 Refresh Token API

class fardel.core.auth.views.**RefreshTokenApi**

URL /api/auth/refresh-token/

post()

- Authorization header containing refresh token is required

Status_code 200

Response

```
{
    "access_token": "access_token"
}
```

2.1.8 Profie API

class fardel.core.auth.views.**ProfileApi**

URL /api/auth/profile/

get()

- Authorization header containing access token is required

Status_code 200

Response

```
{
    "user": UserObject
}
```

put()

- Authorization header containing refresh token is required

Status_code 200

Response

```
{
    "message": "Profile successfully updated"
    "user": UserObject
}
```

2.2 Blog

2.2.1 Objects

1. **Category**

   id ID for the category in database.

   name Name of the category to display.

   posts

2. **Post**

   id ID for the post in database.

   title

   content

   allow_comment

   category

   image

   comments_count

   tags

create_time

update_time

summarized

3. **Tag**

   id ID for the tag in database.

   name Name of the tag.

   frequency

   posts list of PostObjects (conditional).

4. **Comment**

   id ID for the comment in database.

   content Content of the comment.

   create_time (Timestamp) create time of the comment.

   user UserObject *Authentication* if user is login.

   author_mail Author email if user is not login.

   author_name Author name if user is not login.

   replies List of Comment Objects.

All of the listed APIs are restfull and content-type is `application/json`

2.2.2 Post API

class fardel.apps.blog.views.**PostApi**

URL `/api/blog/posts/` and `/api/blog/posts/<post_id>/`

get(*post_id=None*)

Optional url parameter

- post_id

Optional url query string

- page (default: 1)
- per_page (default: 16)

Response If post_id is provided:

```
{ "post": PostObject(with content) }
```

If post_id is not provided:

```
{
    "posts":[list of PostObjects(without content and with␣
↪summarized)]
    "pages": Number of pages
}
```

Errors If post id is not valid:

status_code 404

response

```
{"message":"No post with this id"}
```

2.2.3 Comment API

class fardel.apps.blog.views.**CommentApi**

URL /api/blog/posts/<post_id>/comments/

get (*post_id*)

Optional url query string

- page (default: 1)
- per_page (default: 16)

Response

```
{
    "comments":[list of CommentObjects]
    "pages": Number of pages
}
```

Errors If post_id is not in valid (not a published post or not found):

status_code 404 .. code-block:: python

{“message”:”No post with this id”}

post (*post_id*)

- Authorization token is not required but optional

Required data

- author_name (if you dont use access_token)
- author_email (if you dont use access_token)
- content

Optional data

- parent_comment_id

Response

```
{"message":"Comment successfuly added"}
```

Errors If post_id is not in valid (not a published post or not found):

status_code 404

response

```
{"message":"No post with this id"}
```

If commenter information (user_id or (author_name, author_email)) is not provided:

status_code 422

response

```
{"message":"Author name and Author email are required to␣
↪post a comment  or you need to sign in"}
```

2.2.4 Tag API

class fardel.apps.blog.views.**TagApi**

URL /api/blog/tags/ or /api/blog/tags/<tag_id>/posts/

get(*tag_id=None*)

If tag_id is provided:

response

```
{
    "tag": TagObject(with posts)
}
```

errors If tag_id is not in valid:

status_code 404 .. code-block:: python

{“message”:”No tag found with this id”}

Without tag_id:

response

```
{
    "tags": [list of TagObjects(without posts)]
}
```

2.2.5 Category API

class fardel.apps.blog.views.**CategoryApi**

URL /api/blog/categories/ or /api/blog/categories/<category_id>/posts/

get(*category_id=None*)

If category_id is provided:

optional url query string

- page (default: 1)
- per_page (default: 16)

response

```
{
    "category": CategoryObject(with posts)
}
```

errors If category_id is not in valid:

status_code 404 .. code-block:: python

{“message”:”No category found with this id”}

Without category_id:

response

```
{
    "categories": [list of CategoryObject(without posts)]
}
```

2.3 Media

All files are accessible through this url

URL `/uploads/<path_to_file>`

2.4 Products

2.4.1 Objects

1. **Category**

   id ID for the category in database.

   name Name of the category to display.

   description

   children List of Category objects.

2. **Product**

   id ID for the product in database.

3. **Collection**

   id ID for the collection in database.

2.4.2 Product Category API

class fardel.apps.ecommerce.product.views.**ProductCategory**(***kwargs*)

2.5 Admin User Management

2.5.1 User API

2.5.2 PermissionApi API

2.5.3 Group API

2.6 Admin Blog Management

2.6.1 Post API

2.6.2 Comment API

2.6.3 Tag API

2.6.4 Category API

2.7 Media Management

2.7.1 Objects

1. **File**

   name

   url

   path

2. **Directory**

   name Name of the directory

   path Path of the folder

2.7.2 File API

class fardel.core.panel.views.media.**FileApi**

URL /api/panel/files/

delete()

To delete file

get()

Directory listing of upload folder

User permission required can_see_directory

post()

To create file

2.7.3 Image Album API

class fardel.core.panel.views.media.**ImageAlbumApi**

URL /api/panel/image_album/

delete()

To delete file

get()

Directory listing of upload folder

post()

To create file

To costumize Boghche-Server you can use the documentation below.

CHAPTER 3

Server Documentation

3.1 Authentication

WIP

3.2 Blog

WIP

Python Module Index

f

Index