Know Me API Documentation

Release 0.5.1

Chathan Driehuys

Aug 17, 2017

Contents:

1 Know Me API **1**
1.1 About 1

HTTP Routing Table **27**

CHAPTER 1

Know Me API

API Root https://new-api.knowmetools.com

Documentation http://know-me-api.readthedocs.io/

1.1 About

This is the API behind our Know Me app. It is written in Python using Django and Django Rest Framework.

1.1.1 REST API

The API is available at the following URLs. The endpoints given should be appended to the base URL.

Production https://new-api.knowmetools.com

Staging https://dev.new-api.knowmetools.com

Authorization

This API uses tokens to authenticate and authorize requests. Authentication is done by setting the `Authorization: Token <token content>` on your requests.

API Endpoints

Authentication

Registration

In order to log in, you must have a user account. User accounts can be created by sending a `POST` request to the registration endpoint.

`POST /auth/register/`
Register a new user.

Request JSON Object

- **`email`** (*string*) – The email address to register the user under.
- **`password`** (*string*) – The password to give the new user.
- **`first_name`** (*string*) – The user's first name.
- **`last_name`** (*string*) – The user's last name.

Response JSON Object

- **`id`** (*int*) – The new user's ID.
- **`email`** (*string*) – The new user's email address.
- **`first_name`** (*string*) – The new user's first name.
- **`last_name`** (*string*) – The new user's last name.

Status Codes

- 201 Created – A new user was sucessfully created.
- 400 Bad Request – Bad request; check response for details.
- 403 Forbidden – Authentication credentials were provided. This endpoint requires the user to be unauthenticated.

Login

Logging in will return a token that can be used to authenticate with other endpoints.

`POST /auth/login/`
Log in an existing user.

Request JSON Object

- **`username`** (*string*) – The user's email address.
- **`password`** (*string*) – The user's password.

Response JSON Object

- **`token`** (*string*) – A token the user can use to authenticate with the API.

Status Codes

- 200 OK – User sucessfully authenticated.
- 400 Bad Request – The provided credentials were invalid.

Layer

For real time communications we use Layer. Layer requires an identity token in order to authenticate with their services.

> **Warning:** The validity of the token returned from this endpoint is not guaranteed. If an invalid nonce was provided, the returned token will also be invalid.

POST /auth/layer/
Obtain an identity token for Layer.

Request JSON Object

- **nonce** (*string*) – A nonce from Layer.

Response JSON Object

- **identity_token** (*string*) – An identity token for Layer.

Status Codes

- 201 Created – Identity token successfully created.
- 400 Bad Request – Invalid request. Check the response data for details.

Account

KMUser

A user's know me user contains personal information about the user such as their email or name.

Note: If you are looking to change a user's password, see the *Change Password* endpoint.

GET /account/profile/
Retrieve the requesting user's account information.

Response JSON Object

- **id** (*int*) – The user's ID.
- **email** (*string*) – The user's email address.
- **first_name** (*string*) – The user's first name.
- **last_name** (*string*) – The user's last name.

Status Codes

- 200 OK – The user's information was successfully retrieved.

PATCH /account/profile/
Update the requesting user's information.

Request JSON Object

- **first_name** (*string*) – *(Optional)* The user's new first name.
- **last_name** (*string*) – *(Optional)* The user's new last name.

Response JSON Object

- **id** (*int*) – The user's ID.
- **email** (*string*) – The user's email address.
- **first_name** (*string*) – The user's first name.
- **last_name** (*string*) – The user's last name.

Status Codes

- 200 OK – The user's information was successfully updated.
- 400 Bad Request – Invalid request. Check the response data for details.

Change Password

POST /account/change-password/

Change the password of the currently authenticated user.

Request JSON Object

- **key** (*string*) – *(Optional)* The password reset key authorizing a password change. The key can be obtained from the *password reset view*. Either this field or `old_password` must be given.
- **old_password** (*string*) – *(Optional)* The user's current password. Either this field or `key` must be given.
- **new_password** (*string*) – The user's new password.

Status Codes

- 200 OK – The user's password was successfully changed.
- 400 Bad Request – Invalid request. Check response data for details. This can happen when an invalid `old_password` is provided, or if `new_password` fails the password validation checks.

Reset Password

If a user forgets their password, sending their email address to this endpoint will send them an email with instructions to reset their password.

> **Warning:** Just because a 200 OK response was received does not mean that the provided email address was valid. We can't return any information about the validity of the email without giving away information about which accounts exist.

POST /account/reset-password/

Request a password reset for the account associated with the provided email address.

Request JSON Object

- **email** (*string*) – The email address to send a password reset email to.

Response JSON Object

- **email** (*string*) – The email address that the password reset was sent to.

Status Codes

- 200 OK – A valid email address was received.
- 400 Bad Request – An invalid email address was received.

Email Verification

Before a user can log in, they must have a verified email address. This allows us to contact the user with any account related messages.

Note: We require the user's password to prevent mistyped email addresses from being verified by an unknown user. See #39 for details.

`POST /account/verify-email/`
Verify an email address.

Request JSON Object

- **`key`** (*`string`*) – The confirmation key that was sent to the user's email.
- **`password`** (*`string`*) – The user's password.

Status Codes

- 200 OK – The email address was confirmed.
- 400 Bad Request – Invalid request. Check the response data for details. This can happen if an invalid key was provided, or if the key has expired.

Email Management

Users are allowed to have multiple emails associated with their account. One of these emails is the user's primary address, and receives all notifications. The user can log in with any of their verified emails.

Email List

The email list endpoint allows for listing of a user's email addresses as well as adding new emails.

`GET /account/emails/`
List the requesting user's email addresses.

Response JSON Array of Objects

- **`id`** (*`int`*) – The ID of the email address.
- **`email`** (*`string`*) – The email's address.
- **`verified`** (*`boolean`*) – A boolean indicating if the address has been verified.
- **`verified_action`** (*`int`*) – An integer corresponding to an action to perform when the email is verified. See *Email Verification Actions* for more information.
- **`primary`** (*`boolean`*) – A boolean indicating if the address is the user's primary email.

Status Codes

- 200 OK – The user's email addresses were successfully retrieved.

POST /account/emails/

Add a new email address for the requesting user.

Request JSON Object

- **email** (*string*) – The address of the new email.

Response Headers

- Location – The URL of the created email address' detail view.

Response JSON Object

- **id** (*int*) – The ID of the email address.
- **url** (*string*) – The URL of the email address' detail view.
- **email** (*string*) – The email's address.
- **verified** (*boolean*) – A boolean indicating if the address has been verified.
- **verified_action** (*int*) – An integer corresponding to an action to perform when the email is verified. See *Email Verification Actions* for more information.
- **primary** (*boolean*) – A boolean indicating if the address is the user's primary email.

Status Codes

- 201 Created – The email address was created successfully.
- 400 Bad Request – Invalid request. Check the response data for details.

Email Detail

The email detail endpoint allows for retrieving and updating a specific email address as well as removing email addresses.

GET /account/emails/(int: *id*)/

Get the details of a specific email address.

Response JSON Object

- **id** (*int*) – The ID of the email address.
- **url** (*string*) – The URL of the email address' detail view.
- **email** (*string*) – The email's address.
- **verified** (*boolean*) – A boolean indicating if the address has been verified.
- **verified_action** (*int*) – An integer corresponding to an action to perform when the email is verified. See *Email Verification Actions* for more information.
- **primary** (*boolean*) – A boolean indicating if the address is the user's primary email.

Status Codes

- 200 OK – The email address' details were successfully retrieved.
- 404 Not Found – There is no email address with the given `id` accessible to

the requesting user.

PATCH /account/emails/(int: *id*)/

Update the details of a specific email address.

Request JSON Object

- **primary** (*boolean*) – *(Optional)* A boolean indicating if the specified email address should be the user's new primary email.

Response JSON Object

- **id** (*int*) – The ID of the email address.
- **url** (*string*) – The URL of the email address' detail view.
- **email** (*string*) – The email's address.
- **verified** (*boolean*) – A boolean indicating if the address has been verified.
- **primary** (*boolean*) – A boolean indicating if the address is the user's primary email.

Status Codes

- 200 OK – The email address' details were successfully updated.
- 404 Not Found – There is no email address with the given `id` accessible to the requesting user.

DELETE /account/emails/(int: *id*)/

Delete a specific email address.

Status Codes

- 204 No Content – The email address was successfully deleted.
- 404 Not Found – There is no email address with the given `id` accessible to the requesting user.
- 409 Conflict – The email address is the user's primary address so it could not be deleted.

Email Verification Actions

When an email address is created, an action can be specified to control what happens when the email is verified. This endpoint provides a list of those actions.

GET /account/emails/actions/

Get a list of available verification actions.

Response JSON Array of Objects

- **id** (*int*) – The action's ID.
- **label** (*string*) – The action's label.

Status Codes

- 200 OK – The available actions were successfully retrieved.

Know Me - Emergency

Know Me provides a way for users to store some information about themselves that could be used in case of an emergency.

Emergency Items

Emergency items are similar to profile items, but they are meant to store information for emergency situations.

Emergency Item List

The emergency item list endpoint allows for listing and creation of emergency items.

GET /know-me/users/(**int:** *id*)**/emergency-items/**
List the emergency items for a Know Me user.

Response JSON Array of Objects

- **id** (*int*) – The emergency item's ID.
- **url** (*string*) – The URL of the emergency item's detail view.
- **name** (*string*) – The emergency item's name.
- **description** (*string*) – The emergency item's description. Can be an empty string.
- **media_resource** (*object*) – The media resource associated with the emergency item. Can be `null`.

Status Codes

- 200 OK – The emergency item list was successfully retrieved.
- 404 Not Found – There is no Know Me user with the provided `id` accessible to the requesting user.

POST /know-me/users/(**int:** *id*)**/emergency-items/**
Create a new emergency item.

Request JSON Object

- **name** (*string*) – The emergency item's name.
- **description** (*string*) – *(Optional)* The emergency item's description.
- **media_resource** (*int*) – *(Optional)* The ID of a media resource to attach to the emergency item.

Response Headers

- Location – The URL of the created emergency item's detail view.

Response JSON Object

- **id** (*int*) – The emergency item's ID.
- **url** (*string*) – The URL of the emergency item's detail view.
- **name** (*string*) – The emergency item's name.
- **description** (*string*) – The emergency item's description. This can be an empty string.
- **media_resource** (*object*) – The media resource attached to the item. This can be `null`.

Status Codes

- 200 OK – A new emergency item was successfully created.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no Know Me user with the provided `id` accessible to the requesting user.

Emergency Item Detail

The emergency item detail endpoint allows for retrieving, updating, or deleting of specific emergency items.

GET /know-me/emergency-items/(int: *id*)/
Retrieve a specific emergency item's details.

Response JSON Object

- **id** (*int*) – The emergency item's ID.
- **url** (*string*) – The URL of the emergency item's detail view.
- **name** (*string*) – The emergency item's name.
- **description** (*string*) – The emergency item's description. This can be an empty string.
- **media_resource** (*object*) – The media resource attached to the item. This can be `null`.

Status Codes

- 200 OK – The emergency item's details were successfully retrieved.
- 404 Not Found – There is no emergency item with the provided `id` accessible to the requesting user.

PATCH /know-me/emergency-items/(int: *id*)/
Update a particular emergency item.

Request JSON Object

- **name** (*string*) – *(Optional)* A new name for the item.
- **description** (*string*) – *(Optional)* A new description for the item.
- **media_resource** (*int*) – *(Optional)* The ID of a media resource to attach to the item.

Response JSON Object

- **id** (*int*) – The emergency item's ID.
- **url** (*string*) – The URL of the emergency item's detail view.
- **name** (*string*) – The emergency item's name.
- **description** (*string*) – The emergency item's description. This can be an empty string.
- **media_resource** (*object*) – The media resource attached to the item. This can be `null`.

Status Codes

- 200 OK – The emergency item's details were successfully updated.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no emergency item with the provided `id` accessible to the requesting user.

DELETE /know-me/emergency-items/(int: *id*)/
Delete a particular emergency item.

Status Codes

- 204 No Content – The emergency item was successfully deleted.

- 404 Not Found – There is no emergency item with the provided `id` accessible to the requesting user.

Know Me - Gallery

The gallery is used to store various files associated with a know me user. Media resources (items in the gallery) can be attached to a particular profile item.

Note: Currently there is no way to retrieve media resources that are not attached to a profile item. This will be introduced later as a paid feature.

Gallery View

This endpoint allows for creation of new media resources.

POST /know-me/users/(**int:** *id*)**/gallery/**

Create a new media resource.

Note: Since media resources involve a file, the request be sent with the header `Content-Type: multipart/form-data`.

Parameters

- **id** (*int*) – The ID of the know me user to create a media resource for.

Form Parameters

- **string name** – The name to give the file being uploaded.
- **file file** – The file to upload.

Response Headers

- Location – The URL of the created media resource's detail view.

Response JSON Object

- **id** (*int*) – The ID of the created media resource.
- **url** (*string*) – The URL of the created media resource's detail view.
- **name** (*string*) – The name the media resource was created with.
- **file** (*string*) – The URL of the file attached to the media resource.

Status Codes

- 201 Created – The media resource was succesfully created.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no know me user with the given `id` accessible to the requesting user.

Media Resource View

This endpoint allows for retrieving and updating a specific media resource's information.

GET /know-me/media-resources/(int: *id*)/
Get the information of a specific media resource.

Parameters

- **id** (*int*) – The ID of the media resource to retrieve.

Response JSON Object

- **id** (*int*) – The ID of the media resource.
- **url** (*string*) – The URL of the media resource's detail view.
- **name** (*string*) – The name of the media resource.
- **file** (*string*) – The URL of the file attached to the media resource.

Status Codes

- 200 OK – The media resource's information was succesfully retrieved.
- 404 Not Found – There is no media resource with the given `id` accessible to the requesting user.

PATCH /know-me/media-resources/(int: *id*)/
Update a specific media resource's information.

Parameters

- **id** (*int*) – The ID of the media resource to update.

<form string name *(Optional)* A new name for the media resource.

<form file file *(Optional)* A new file to associate with the media resource.

Status Codes

- 200 OK – The media resource was succesfully updated.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no media resource with the given `id` accessible to the requesting user.

Know Me - KMUser

These endpoints provide data for the Know Me app.

KMUsers

KMUsers are the basis of Know Me. They contain organized sets of information about a specific user.

KMUser List

GET /know-me/users/
Get the list of know me users that the requesting user has access to.

Response JSON Array of Objects

- **id** (*int*) – The know me user's ID.
- **url** (*string*) – The URL of the know me user's detail view.
- **name** (*string*) – The name of the know me user.
- **quote** (*string*) – A quote from the user who owns the know me user.

Status Codes

- 200 OK – The request was successful.

POST /know-me/users/

Create a new know me user for the user making the request.

Request JSON Object

- **name** (*string*) – A name for the know me user.
- **quote** (*string*) – A quote from the user.

Response Headers

- Location – The URL of the created know me user's detail view.

Response JSON Object

- **id** (*int*) – The know me user's ID.
- **url** (*string*) – The URL of the know me user's detail view.
- **name** (*string*) – The name of the know me user.
- **quote** (*string*) – A quote from the user who owns the know me user.

Status Codes

- 201 Created – The new know me user was successfully created.
- 400 Bad Request – Invalid request. Check the response data for details.

Note: Currently, a user may only have one know me user.

KMUser Details

GET /know-me/profiles/(int: *id*)/

Get the details of a specific know me user.

Parameters

- **id** – The ID of the know me user to get.

Response JSON Object

- **id** (*int*) – The know me user's ID.
- **url** (*string*) – The URL of the know me user's detail view.
- **name** (*string*) – The name of the know me user.
- **quote** (*string*) – A quote from the user who owns the know me user.
- **emergency_items_url** (*string*) – The URL of the user's emergency item list.

- **gallery_url** (*string*) – The URL of the know me user's gallery.
- **profiles_url** (*string*) – The URL of the know me user's profile list.
- **profiles** (*array*) – A list of the profiles contained in the know me user.

Status Codes

- 200 OK – The know me user's details were retrieved succesfully.
- 404 Not Found – There is no know me user with the given *id* accessible to the requesting user.

PATCH /know-me/profiles/(int: *id*)/

Update a specific know me user's details.

Parameters

- **id** – The ID of the know me user to update.

Request JSON Object

- **name** (*string*) – *(Optional)* The know me user's new name.
- **quote** (*string*) – *(Optional)* The know me user's new quote.

Response JSON Object

- **id** (*int*) – The know me user's ID.
- **url** (*string*) – The URL of the know me user's detail view.
- **name** (*string*) – The name of the know me user.
- **quote** (*string*) – A quote from the user who owns the know me user.
- **emergency_items_url** (*string*) – The URL of the know me user's emergency item list.
- **gallery_url** (*string*) – The URL of the know me user's gallery.
- **profiles_url** (*string*) – The URL of the know me user's profile list.
- **profiles** (*array*) – A list of the profiles contained in the know me user.

Status Codes

- 200 OK – The know me user's details were succesfully updated.
- 400 Bad Request – The update failed. Check the response data for details.

Profiles

Profiles are the next step down in a know me user. They contain information targeted towards a profile of people.

Profile List

The profile list endpoint allows for listing of a know me user's profiles as well as creation of new profiles.

GET /know-me/users/(int: *id*)/profiles/

List the profiles in a particular know me user.

Parameters

- **id** (*int*) – The ID of the know me user to fetch the profiles of.

Response JSON Array of Objects

- **id** (*int*) – The ID of the profile.
- **url** (*string*) – The URL of the profile's detail view.
- **name** (*string*) – The name of the profile.
- **is_default** (*boolean*) – A boolean representing if the profile is the default for its know me user.

Status Codes

- 200 OK – The know me user's profiles were retrieved succesfully.
- 404 Not Found – No know me user with the given *id* was found.

POST /know-me/users/(int: id)/profiles/

Create a new profile for the given know me user.

Parameters

- **id** (*int*) – The ID of the know me user to create a profile for.

Request JSON Object

- **name** (*string*) – The name of the profile.
- **is_default** (*boolean*) – *(Optional)* A boolean determining if the profile will be the default profile for the know me user. Defaults to `false`.

Response Headers

- Location – The URL of the created profile's detail view.

Response JSON Object

- **id** (*int*) – The ID of the profile.
- **url** (*string*) – The URL of the profile's detail view.
- **name** (*string*) – The name of the profile.
- **is_default** (*boolean*) – A boolean representing if the profile is the default for its know me user.

Status Codes

- 201 Created – The profile was successfully created.
- 400 Bad Request – Invalid request. Check the response data for details.

Profile Detail

The profile detail endpoint allows for viewing and updating a profile's information.

GET /know-me/profiles/(int: id)/

Get the details of a particular profile.

Parameters

- **id** (*int*) – The ID of the profile to fetch.

Response JSON Object

- **id** (*int*) – The ID of the profile.

- **url** (*string*) – The URL of the profile's detail view.
- **name** (*string*) – The name of the profile.
- **is_default** (*boolean*) – A boolean representing if the profile is the default for its know me user.
- **topics_url** (*string*) – The URL of the profile's topic list.
- **topics** (*array*) – A list of the profile topics contained in the profile.

Status Codes

- 200 OK – The profile's details were retrieved succesfully.
- 404 Not Found – There is no profile with the given `id` accessible to the requesting user.

PATCH /know-me/profiles/(int: *id*)/

Update a specific profile's information.

Parameters

- **id** (*int*) – The ID of the profile to update.

Request JSON Object

- **name** (*string*) – *(Optional)* A new name for the profile.
- **is_default** (*boolean*) – *(Optional)* The new `is_default` status for the profile.

Response JSON Object

- **id** (*int*) – The ID of the profile.
- **url** (*string*) – The URL of the profile's detail view.
- **name** (*string*) – The name of the profile.
- **is_default** (*boolean*) – A boolean representing if the profile is the default for its know me user.
- **topics_url** (*string*) – The URL of the profile's topic list.
- **topics** (*array*) – A list of the know me user topics contained in the profile.

Status Codes

- 200 OK – The profile's information was succesfully updated.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no profile with the given `id` accessible to the requesting user.

Profile Topics

Profile topics hold specific categories of information for a profile.

Profile Topic List

GET /know-me/profiles/(int: *id*)/topics/

List the topics in a particular profile.

Parameters

- **id** (*int*) – The ID of the profile to fetch the topics of.

Response JSON Array of Objects

- **id** (*int*) – The ID of the topic.
- **url** (*string*) – The URL of the topic's detail view.
- **name** (*string*) – The name of the topic.
- **topic_type** (*int*) – An integer representing the type of the topic.
- **items_url** (*string*) – The URL of the topic's item list.
- **items** (*array*) – The items contained in the topic.

Status Codes

- 200 OK – The profile topic list was succesfully retrieved.
- 404 Not Found – There is no profile with the given `id` accessible to the requesting user.

POST /know-me/profiles/(int: *id*)/topics/

Create a new profile topic in a particular profile.

Parameters

- **id** (*int*) – The ID of the profile to create a topic for.

Response JSON Array of Objects

- **topics_url** (*string*) – The URL of the given topic's list.
- **topics** (*object*) – An object containing the profile's topic.

Request JSON Object

- **name** (*string*) – A name for the topic.
- **topic_type** (*int*) – An integer representing which type of topic to create.

Response Headers

- Location – The URL of the created topic's detail view.

Response JSON Object

- **id** (*int*) – The ID of the topic.
- **url** (*string*) – The URL of the topic's detail view.
- **name** (*string*) – The name of the topic.
- **topic_type** (*int*) – An integer representing the type of topic.
- **items_url** (*string*) – The URL of the topic's item list.
- **items** (*array*) – The items contained in the topic.

Status Codes

- 201 Created – The profile topic was succesfully created.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no profile with the given `id` accessible to the requesting user.

Profile Topic Detail

This endpoint allows for viewing and updating a specific profile topic's information.

GET /know-me/topics/(int: *id*)/
Get a specific profile topic's information.

Parameters

- **id** (*int*) – The ID of the profile topic to fetch.

Response JSON Object

- **id** (*int*) – The ID of the topic.
- **url** (*string*) – The URL of the topic's detail view.
- **name** (*string*) – The name of the topic.
- **topic_type** (*int*) – An integer representing the type of topic.
- **items_url** (*string*) – The URL of the topic's item list.
- **items** (*array*) – The items contained in the topic.

Status Codes

- 200 OK – The profile topic's information was succesfully retrieved.
- 404 Not Found – There is no profile topic with the given `id` accessible to the requesting user.

PATCH /know-me/topics/(int: *id*)/
Update a specific profile topic's details.

Parameters

- **id** (*int*) – The ID of the topic to update.

Request JSON Object

- **name** (*string*) – *(Optional)* A new name for the topic.
- **topic_type** (*int*) – *(Optional)* The topic's new type, as an integer.

Response JSON Object

- **id** (*int*) – The ID of the topic.
- **url** (*string*) – The URL of the topic's detail view.
- **name** (*string*) – The name of the topic.
- **topic_type** (*int*) – An integer representing the type of topic.
- **items_url** (*string*) – The URL of the topic's item list.
- **items** (*array*) – The items contained in the topic.

Status Codes

- 200 OK – The profile topic's information was succesfully updated.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no profile topic with the given `id` accessible to the requesting user.

Profile Items

Profile items contain specific pieces of the information in a profile topic.

Profile Item List

This endpoint allows for listing the items in a profile topic and adding new items to the topic.

GET /know-me/topics/(**int:** *id*)**/items/**
List the items in a profile topic.

Parameters

- **id** (*int*) – The ID of the profile topic to fetch the items for.

Response JSON Array of Objects

- **id** (*int*) – The ID of the item.
- **url** (*string*) – The URL of the item's detail view.
- **name** (*string*) – The name of the item.
- **image_content** (*object*) – An object containing the item's image content. May be `null`.
- **list_content** (*object*) – An object containing the item's list content. May be ‘‘null‘.

Status Codes

- 200 OK – The profile item list was succesfully retrieved.
- 404 Not Found – There is no profile topic with the given `id` accessible to the requesting user.

POST /know-me/topics/(**int:** *id*)**/items/**
Create a new profile item in a particular topic.

Parameters

- **id** (*int*) – The ID of the profile topic to create an item in.

Request JSON Object

- **name** (*string*) – The name of the item.
- **image_content** (*object*) – An object containing the item's image content. Mutually exclusive with `list_content`.
- **list_content** (*object*) – An object containing the item's list content. Mutually exclusive with `image_content`.

Response Headers

- Location – The URL of the created item's detail view.

Response JSON Object

- **id** (*int*) – The ID of the item.
- **url** (*string*) – The URL of the item's detail view.
- **name** (*string*) – The name of the item.
- **image_content** (*object*) – An object containing the item's image content. May be `null`.

- **list_content** (*object*) – An object containing the item's list content. May be ‘‘null‘.

Status Codes

- 201 Created – The profile item was succesfully created.
- 400 Bad Request – Invalid request. Check the response data for details.
- 404 Not Found – There is no profile topic with the given `id` accessible to the requesting user.

Profile Item Detail

This endpoint allows for retrieving and updating a specific profile item's information.

GET /know-me/items/(int: *id*)/
Retrieve a specific profile item's information.

Parameters

- **id** (*int*) – The ID of the profile item to fetch.

Response JSON Object

- **id** (*int*) – The ID of the item.
- **url** (*string*) – The URL of the item's detail view.
- **name** (*string*) – The name of the item.
- **image_content** (*object*) – An object containing the item's image content. May be `null`.
- **list_content** (*object*) – An object containing the item's list content. May be ‘‘null‘.

Status Codes

- 200 OK – The profile item's information was succesfully retrieved.
- 404 Not Found – There is no profile item with the given `id` accessible to the requesting user.

PATCH /know-me/items/(int: *id*)/
Update a specific profile item's information.

Parameters

- **id** (*int*) – The ID of the profile item to update.

Request JSON Object

- **name** (*string*) – *(Optional)* A new name for the item.

Response JSON Object

- **image_content** (*object*) – *(Optional)* An object containing the item's image content. May be `null`.
- **list_content** (*object*) – *(Optional)* An object containing the item's list content. May be ‘‘null‘.
- **id** (*int*) – The ID of the item.
- **url** (*string*) – The URL of the item's detail view.
- **name** (*string*) – The name of the item.
- **image_content** – An object containing the item's image content. May be `null`.

- **list_content** – An object containing the item's list content. May be ‘‘null‘.

Status Codes

- 200 OK – The profile item's information was succesfully updated.
- 404 Not Found – There is no profile item with the given `id` accessible to the requesting user.

1.1.2 Development

The API is built with Python using Django and Django Rest Framework.

Recommended Development Environment

If you are comfortable with setting up a python development environment and cloning the project, feel free to skip to the *development environment overview*.

Prerequisites

We use git for version control and generally follow the development model laid out here. If you are looking for a tool to assist in following this model, we recommend git-flow, a tool made by the same people that created the development model.

Since this is a Python project, we recommend using a virtualenv to manage the environment for the project. If you want to simplify the management of these virtual environments, we recommend using virtualenvwrapper.

This project runs on Python 3.4, since that's the Python version available on Elastic Beanstalk, our hosting platform. Ideally your development environment also has Python 3.4 installed. If not, you can find it here

Project Setup

Note: The following commands assume the `virtualenvwrapper` package is installed.

Before cloning the environment, create a virtual environment for the project's dependencies:

```
$ mkproject --python=python3.4 km-api
$ workon km-api
```

You can now clone the project. Since the directory will have already been created, we initialize an empty git repository and then add the project as a remote:

```
$ git init
$ git remote add origin https://github.com/knowmetools/km-api
$ git pull origin develop
```

If you installed the `git-flow` extension, you can now setup the repository to use it:

```
$ git flow init -d
```

Finally, install the project requirements appropriate for what you need. The `test` requirements should cover what you need. If all you want to do is run the project locally, the `base` requirements are all you need.:

```
$ pip install -r requirements/[base|test].txt
```

Linting

We use flake8 to lint our code, which is a tool for checking compliance with python's style guide: pep8. To lint the source code, run flake8 from the project root:

```
$ flake8
```

If you want to run the linter on every commit, which is useful because our CI tool fails a build with linting errors, you can install flake8's git hook:

```
$ flake8 --install-hook git
$ git config flake8.lazy true
$ git config flake8.strict true
```

The configuration options ensure that only the code being committed is linted, and that linting errors will stop the commit process.

Dev Environment Overview

If you have not yet cloned the repository, do so and install the requirements:

```
$ git clone https://github.com/knowmetools/km-api
$ cd km-api
$ pip install -r requirements/base.txt
```

Local Dev Server

The development server can be run using the following command:

```
$ km_api/manage.py runserver
```

Running Tests

Tests are run with pytest. To run the tests, make sure the requirements are installed and run the tests:

```
$ pip install -r requirements/test.txt
$ pytest km_api/
```

Building Docs

We use sphinx for building documentation, and the docs are automatically published using ReadTheDocs. If you want to build the docs locally, install the requirements and run the build command:

```
$ pip install -r requirements/docs.txt
$ cd docs
$ make html
```

1.1.3 Deployment

Deployment is handled through AWS' Elastic Beanstalk service. Tagged releases and commits on the `develop` branch are automatically deployed to production and staging, respectively.

Environment Variables

The application uses the following environment variables. These can be set from the Elastic Beanstalk interface.

ADMIN_EMAIL The email address to use for the admin account.

ADMIN_PASSWORD The password to use for the admin account.

ALLOWED_HOSTS (=[ ]) A comma separated list of URLs that the app is accessible from.

AWS_REGION (=us-east-1) The region the project's AWS resources are running in.

DEBUG (=False) Set to `True` (case insensitive) to enable Django's debug mode.

EMAIL_CONFIRMATION_EXPIRATION_DAYS (=1) An integer specifying the number of days an email confirmation is valid for.

EMAIL_CONFIRMATION_LINK_TEMPLATE (=https://example.com/confirm-email?key={key}) A template for the URL a user should visit to validate their email. The value `{key}` in the template string will be replaced with the confirmation key.

LAYER_IDENTITY_EXPIRATION (=300) The expiration time of each Layer identity token in seconds. See Layer's Identity Token documentation for more information.

LAYER_KEY_ID The ID of the key located at `LAYER_RSA_KEY_FILE_PATH`. This can be found in Layer's organization dashboard. It should have the format `layer:///keys/<key-content>`.

LAYER_PROVIDER_ID The provider ID of the Layer organization. This can be found in Layer's organization dashboard. It should have the format `layer:///providers/<provider-id>`.

LAYER_RSA_KEY_FILE_PATH (=/etc/km-api/certs/layer-dev.pem) The path to the RSA key used to encode the identity tokens for Layer.

MAILCHIMP_API_KEY (='') The API key to use when using the MailChimp API.

MAILCHIMP_ENABLED (=False) Set to `True` (case insensitive) to enable syncing of user data to a MailChimp list. Requires `MAILCHIMP_API_KEY` and `MAILCHIMP_LIST_ID` to be set.

MAILCHIMP_LIST_ID (='') The ID of the MailChimp list to sync users to. Can be found under the list's "Settings" menu in "List name and campaign defaults".

PASSWORD_RESET_EXPIRATION_HOURS (=1) The number of hours a password reset's key is valid for.

PASSWORD_RESET_LINK_TEMPLATE (=https://example.com/change-password/?key={key}) A template for the URL a user should visit to complete the password reset process. The value `{key}` in the template string will be replaced with the password reset key.

RDS_DB_NAME The database's name.

RDS_HOSTNAME: The hostname of the database.

RDS_PASSWORD: The password to connect to the database with.

RDS_PORT: The port to connect to the database on. This is usually 5432.

RDS_USERNAME: The username to connect to the database with.

SECRET_KEY The secret key to use. This should be a long random string. See the documentation for details.

SENTRY_DSN The DSN to use for sentry logging. See the documentation for details.

SENTRY_ENVIRONMENT (=staging) The environment to use when logging errors to sentry. This allows for differentiating between production and staging errors. For simplicity, this should be either `staging` or `production`.

STATIC_BUCKET The name of the S3 bucket to store static and media files in. The IAM role that the webservers use must have access to this bucket. This bucket must be in the `us-east-1` region.

Database Provisioning

Database provisioning is handled with Ansible using the playbooks in the `deploy` directory.

Prerequisites

To run the playbook, you need Ansible installed as well as some helper python packages:

```
$ pip install ansible boto psycopg2
```

Note: It is assumed that these packages are installed for the system-wide python install. If you would like to run ansible with an arbitrary python interpreter, pass in the `--ansible-python-interpreter=<path to python>` flag to any ansible command.

In order to run the playbook, you must also have valid credentials. AWS credentials must either be set as environment variables or passed to Ansible with the `--extra-vars` flag. Finally you must have the Ansible vault password.

Running The Playbook

To run the playbook and provision a database:

```
$ ansible-playbook deploy/deploy.yml
```

If you want to target the production environment:

```
$ ansible-playbook --extra-vars '"env"="prod"' deploy/deploy.yml
```

Configuring the Application

After running the playbook, you **must** update the application configuration in Elastic Beanstalk. Specifically, you must ensure that the `RDS_*` settings are correct. If the database was recreated, you must also ensure that the migrations have been run. The simplest way to do that is to trigger a deployment:

```
$ eb deploy <env-name>
```

1.1.4 Changelog

v0.5.1

Bug Fixes

- #143: Fix missing field on emergency contact admin page.

v0.5

This release was focused on renaming the components of a Know Me user's profile. As a result of this renaming, this release will break all existing data related to Know Me. This change also caused several endpoints to be renamed. The most relevant issue here is #65.

- `/know-me/gallery-items/*` to `/know-me/media-resources/*`
- `/know-me/profiles/*` to `/know-me/users/*`
- `/know-me/rows/*` to `/know-me/topics/*`

Breaking Changes

- #66: Remove grouped and paged row types.
- #85: Separated content for different types of profile items into different models. This means the data (other than name) from existing profile items is lost.

Features

- #67: Add emergency contacts for Know Me users.
- #68#100#112: Add list-type profile items.
- #79#91#96: Add ability for users to manage their emergency items.

Bug Fixes

- #70: Fixed regression in the error returned when attempting to log in with an unverified email address.
- #75: Fix parsing of JSON requests.

v0.4

Breaking Changes

- #27: Move user profile view from `/auth/profile/` to `/account/profile`.
- #36#54: Emails must be verified before being able to log in.
- #42: The user model was moved to the `account` app. This requires dropping any existing databases.

Features

- #28: Users can change their password.
- #34: Users can now request a password reset by email.
- #47: Allow users to manage their email addresses. They can now add/remove addresses and switch which one is the primary.
- #50: Users receive a notification when an email is added to their account.

Miscellaneous

- #41: Users can be authenticated by passing an `email` rather than a `username` to Django's `authenticate` function.
- #46: An admin user is created when the project is deployed.
- #52: Developers are no longer required to have a local settings file.

v0.3

Features

- #29#30#31: Automatically sync user info to a MailChimp list.

Miscellaneous

- #32: Ignore reports about disallowed hosts.

v0.2

Breaking Changes

- #18: Flattened URL structure.
- #21: Moved Layer authentication to the `/auth/layer/` endpoint.

Features

- #12: Add logging in production.
- #14: Refactor permissions implementation using `dry-rest-permissions` package.
- #19#20: Add documentation.

Bug Fixes

- #9: Ensure passwords are validated.

1.1.5 Indices and tables

- genindex
- modindex
- search

HTTP Routing Table

/account

GET /account/emails/, 5
GET /account/emails/(int:id)/, 6
GET /account/emails/actions/, 7
GET /account/profile/, 3
POST /account/change-password/, 4
POST /account/emails/, 5
POST /account/reset-password/, 4
POST /account/verify-email/, 5
DELETE /account/emails/(int:id)/, 7
PATCH /account/emails/(int:id)/, 6
PATCH /account/profile/, 3

/auth

POST /auth/layer/, 3
POST /auth/login/, 2
POST /auth/register/, 2

/know-me

GET /know-me/emergency-items/(int:id)/, 9
GET /know-me/items/(int:id)/, 19
GET /know-me/media-resources/(int:id)/, 11
GET /know-me/profiles/(int:id)/, 14
GET /know-me/profiles/(int:id)/topics/, 15
GET /know-me/topics/(int:id)/, 17
GET /know-me/topics/(int:id)/items/, 18
GET /know-me/users/, 11
GET /know-me/users/(int:id)/emergency-items/, 8
GET /know-me/users/(int:id)/profiles/, 13
POST /know-me/profiles/(int:id)/topics/, 16
POST /know-me/topics/(int:id)/items/, 18
POST /know-me/users/, 12
POST /know-me/users/(int:id)/emergency-items/, 8
POST /know-me/users/(int:id)/gallery/, 10
POST /know-me/users/(int:id)/profiles/, 14
DELETE /know-me/emergency-items/(int:id)/, 9
PATCH /know-me/emergency-items/(int:id)/, 9
PATCH /know-me/items/(int:id)/, 19
PATCH /know-me/media-resources/(int:id)/, 11
PATCH /know-me/profiles/(int:id)/, 15
PATCH /know-me/topics/(int:id)/, 17