Contents:

Welcome to HatecrimeDB documentation!¶

1. High-Level Design (Architecture)¶

1.1. Operation system¶

The project should run on Linux system. Prefered OS are Ubuntu LTS (https://wiki.ubuntu.com/LTS) or latest stable CentOS (https://www.centos.org/). While it may be possible to run the system under the Windows server, it has been tested for compability.

1.2. Execution environment¶

The system executes in Python 3.6 (https://www.python.org/) environment.

1.3. Web server¶

The system can run behind both Apache (https://httpd.apache.org/) and Nginx server (https://nginx.org/en/).

1.4. Database engine¶

The database backend for data persistence is powered by PostgreSQL v9.6. Official page: https://www.postgresql.org/

1.5. Architectural pattern¶

The system implements MVC (Model-View-Controller https://en.wikipedia.org/wiki/Model-view-controller) pattern based on Django's implementation called MTV (Model Template View), which is basically the same as MVC but differs in terminology: View is Template and Controller is View in Django terms.

1.6. Used frameworks¶

- Web backend and frontend is powered by Django v1.11 which is a python language web framework. Official page: https://www.djangoproject.com/
- HatecrimeDB Public API Integration module for Drupal part of the system is powered by Vue.JS v2.3 which is a javascript web framework that doesn't require any server side services due to its client nature (executes solely in user's browser). Official page: https://vuejs.org/.

1.7. 3rd party Django / Python extensions used¶

While the system is built on top of Django, there are multiple third party extensions were used to extend the functionality and help custom development.

A list of extensions can be found in `requirements.txt` file. You can also see a list of enabled extensions in `config.settings.base.THIRD_PARTY_APPS` list (see `config/settings/base.py` file).

Here's a list of 3rd party extensions:

rest_framework

Django Rest Framework (http://www.django-rest-framework.org/) is a powerful and flexible toolkit for building Web APIs. DRF powers *hatecrimedb.api* module which is used to expose public_data api to be consumed by Drupal.

crispy_forms

Django Crispy Forms (http://django-crispy-forms.readthedocs.io/en/latest/) provides a |crispy filter and {% crispy %} tag that will let you control the rendering behavior of Django forms in a very elegant and DRY way. We use it for forms styling.

allauth

Django AllAuth (https://github.com/pennersr/django-allauth) integrated set of Django applications addressing authentication, registration, account management. Our user model and authentication relies upon this module.

bootstrap3

Django Bootstrap (https://github.com/dyve/django-bootstrap3) bootstrap 3 integration with Django. Ee use Bootstrap theme v3 (https://getbootstrap.com/) as our base theme for the frontend.

bootstrapform

Django bootstrap form (https://django-bootstrap-form.readthedocs.io/en/latest/) a simple Django template tag to work with twitter bootstrap. We use it to add bootstrap style support to Django forms.

django_fsm

Django Finite State Machine (https://github.com/kmmbvnr/django-fsm) adds simple declarative states management for django models. It's responsible for states of *Incidents* and *Reports*.

djangoformsetjs

Django Formset JS (https://bitbucket.org/tim_heap/django-formset-js) adds Javascript support to formsets. We've forked this module to fix bugs and add additional functionality. The fork can be found under *vendors/djangoformsetjs* directory.

dal / dal_select2

Django Autocomplete Light (https://github.com/yourlabs/django-autocomplete-light) a fresh approach to autocomplete implementations. It powers autocomplete input and select boxes in forms on the frontend.

django_filters

Django Filter (https://github.com/carltongibson/django-filter) is a reusable Django application allowing users to declaratively add dynamic QuerySet filtering from URL parameters. It powers search and filters on the frontend.

actstream

Action Streams (http://django-activity-stream.readthedocs.io/en/latest/streams.html) generates generic activity streams from the actions on your site. Users can follow any actors' activities for personalized streams. We use it for logging model history.

debug_permissions

Django Debug Permissions (https://github.com/timonweb/django-debug-permissions) returns a list of all user permissions available in the system. It's a dev module.

dbbackup

Django Database Backup (https://github.com/django-dbbackup/django-dbbackup) management commands to help backup and restore your project database and media files.

dbbackup_ui

Django Database Backup UI (https://github.com/timonweb/django-dbbackup-ui) backup database and media files via Django admin interface. Adds ability to download backups from admin interface. Accessible via http://hatecrimedb.odihr.pl/backdoor/backups/backup-database-and-media/.

file_resubmit

Django File Resubmit (https://github.com/un1t/django-file-resubmit) preserves uploaded files on a form's validation fail. We maintain our own fork of this module under *vendors/file_resubmit* directory.

flexible_date

A custom field that can store a date with flexible granularity (i.e. only year, year+month, or full date). We maintain our own fork of this module under *vendors/flexible_date* directory.

corsheaders

Django Cors Headers (https://github.com/ottoyiu/django-cors-headers) a Django App that adds CORS (Cross-Origin Resource Sharing) headers to responses. We use it for hatecrimedb.api rest endpoints.

2. Project layout¶

2.1. Project directories¶

The project's source code can be found under */code* directory.

There are three main directories inside of */code*:

- `config` - holds Django configuration and main wsgi.py file.
- `hatecrimedb`- contains main application code of the system
- `vendors` - contains forked 3rd party apps that have been fixed / improved to meet project needs.

2.2. Source code organization¶

2.2.1. config directory contents¶

Under config directory you can find three main elements:

- `urls.py` - is the main router of the app, all app urls start here,
- `wsgi.py` - a Django wsgi entrypoint,
- `settings` - a directory that holds app configuration splitted into `base.py` (common app settings), `local.py` (dev settings) and `production.py` (production settings).

2.2.2. hatecrimedb directory contents¶

Main part of the app resides under *hatecrimedb* directory. Django embraces a concept of "Apps" and promotes splitting the app into smaller chunks.

A list of apps available under *hatecrimedb* and their functions can be found under: Functionality, models, controllers and templates broken down by apps.

There are also *non-app* directories:

- `media` - storage for all user uploaded content.
- `static` - storage for javascript and css assets required by apps.
- `templates` - there you can find global app templates and template overrides for 3rd party apps.

2.2.3. vendors directory contents¶

While working on the app, we've forked several 3rd party apps to make them meet our requirements.

A list of apps available under *vendors* can be found under: Information about forks of 3rd party apps.

3. Database Structure¶

System's database schema is driven by Django models. Every model has a corresponding database table where it stores data.

You can read more information about apps and their corresponding models in this section: Functionality, models, controllers and templates broken down by apps.

3.1. Database schema¶

On the figure below you can see an overview of the database schema and relations between tables:

_images/models.png

Click on the image to see full version. If you read this doc in printed, please find this schema in Appendix I.

The following schema is generated by Django via command `python manage.py graph_models`.

3.2. Database indexes¶

You can see indexes in bold on the database schema image above.

In general:

- Every model has index on the primary key which is `ID`.
- Every model has index on every foreign key.

3.3. Database triggers¶

We don't use any built-in database triggers. All events that alter database data happen at the application level.

4. Backend services¶

System's database schema is driven by Django models. Every model has a corresponding database table where it stores data.

You can read more information about apps and their corresponding models in this section: Functionality, models, controllers and templates broken down by apps.

4.1. Cache engines¶

The system utilizes Django's cache framework (https://docs.djangoproject.com/en/1.11/topics/cache/) with a simple in-memory cache backend (`django.core.cache.backends.locmem.LocMemCache`). Cache settings can be found in `config.settings.base.CACHES`.

The backend can be easily changed to Redis or Memcache cache should the better perfromance be needed.

4.2. Cron jobs¶

There's a single cron job configured for the system that runs a scheduled backup task powered by hatecrime.dbbackup_email module.

Below you can see an example of the job:

```
5 23 * * * export DJANGO_SETTINGS_MODULE="config.settings.production" && /var/www/vhosts/hatecrimedb.odihr.pl/.virtualenvs/hatecrimedb_live/bin/pythor
```

4.3. Mail server¶

The system uses mail server to send emails out. Currently it uses a 3rd party email server backed by Google Gmail. The email backend settings can be changed using the following variables:

- `config.settings.production.EMAIL_HOST`, smpt server address, currently set to *smtp.gmail.com*.
- `config.settings.production.EMAIL_HOST_USER`, smtp user, currently set to gmail account.
- `config.settings.production.EMAIL_HOST_PASSWORD`, smtp user password.

4.4. Other backend tasks¶

There are no other special backend tasks or services used by the system.

5. Functionality, models, controllers and templates broken down by apps¶

Django embraces a concept of "Apps" and promotes splitting the app into smaller chunks.

Below you can find a list of custom app that were developed to meet project needs.

5.1. hatecrimedb.api (api)¶

Exposes publicly available api that exports information to be consumed by Drupal and other systems (if needed).

Filters, aggregates and prepares information to be consumed by 3rd party clients.

This app heavily depends on Django Rest Framework (http://www.django-rest-framework.org/).

5.1.1. Views (Controllers)¶

There are two viewsets based on Django Rest Framework (www.django-rest-framework.org) Viewsets:

class `hatecrimedb.api.views.IncidentViewSet`(***kwargs*)¶

A viewset that exposes various endpoints (/public-data/incidents/*) for incidents data.

`by_bm`(*request*, **args*, ***kwargs*)¶

Returns incidents grouped by bias motivation.

Example HTTP request – /public-data/incidents/by-bm/?country_name=Norway&year=2016&limit=20&ordering=date

`list`(*request*, **args*, ***kwargs*)¶

Returns a list of incidents.

Example HTTP request – /public-data/incidents/

`summary_bm_by_country`(*request*, **args*, ***kwargs*)¶

Returns summary by country.

Example HTTP request – /public-data/incidents/summary-bm-by-country/?bias_motivations=4&year=2016

`summary_bm_by_type_of_crime`(*request*, **args*, ***kwargs*)¶

Returns summary by type of crime.

Example HTTP request – /public-data/incidents/summary-bm-by-type-of-crime/?country_name=Norway&year=2016

class `hatecrimedb.api.views.BiasMotivationViewSet`(***kwargs*)¶

`list`(*request*, **args*, ***kwargs*)¶

Returns a list of bias motivations.

Example HTTP request – /public-data/bias-motivations/

5.2. hatecrimedb.common (common)¶

This app holds a set of models, views and templates that are used by other apps of the system.

5.2.1. Models¶

class `hatecrimedb.common.models.Auditable`(**args*, ***kwargs*)¶

> Mixin that combines capabilities of `AuthorLastEditorMixin` and `TimeStampedModel` and makes models that inherit it autitable as a result (stores information about who created the model, who was the last person edited it, and model creation/update date and time).

class `hatecrimedb.common.models.AuthorLastEditorMixin`(**args*, ***kwargs*)¶

> Provides `author` and `last_editor` foreign keys to be used by other models.

class `hatecrimedb.common.models.AuthorMixin`(**args*, ***kwargs*)¶

> Provides `author` foreign key to be used by other models.

class `hatecrimedb.common.models.Comment`(**args*, ***kwargs*)¶

> A base model that provides basic comment functionality.
>
> `save`(**args*, ***kwargs*)¶
>
> > Saves the current instance. Override this in a subclass if you want to control the saving process.
> >
> > The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class `hatecrimedb.common.models.Country`(**args*, ***kwargs*)¶

> Stores information about countries available for other models.
>
> *exception* `DoesNotExist`¶
>
> *exception* `MultipleObjectsReturned`¶

class `hatecrimedb.common.models.File`(**args*, ***kwargs*)¶

> A base model that provides basic file uploading functionality.
>
> `save`(**args*, ***kwargs*)¶
>
> > Saves the current instance. Override this in a subclass if you want to control the saving process.
> >
> > The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class `hatecrimedb.common.models.Language`(**args*, ***kwargs*)¶

> Stores information about languages available for other models.
>
> *exception* `DoesNotExist`¶
>
> *exception* `MultipleObjectsReturned`¶

class `hatecrimedb.common.models.LastEditorMixin`(**args*, ***kwargs*)¶

> Provides `last_editor` foreign key to be used by other models.

class `hatecrimedb.common.models.Organization`(**args*, ***kwargs*)¶

> Stores information about organizations available for other models.
>
> *exception* `DoesNotExist`¶
>
> *exception* `MultipleObjectsReturned`¶

class `hatecrimedb.common.models.OrganizationComment`(**args*, ***kwargs*)¶

> Stores comments for `Organization`.
>
> *exception* `DoesNotExist`¶
>
> *exception* `MultipleObjectsReturned`¶

class `hatecrimedb.common.models.TimeStampedModel`(**args*, ***kwargs*)¶

> An abstract base class model that provides self-updating `created` and `modified` fields.

5.2.2. Views (Controllers)¶

class `hatecrimedb.common.views.OrganizationList`(***kwargs*)¶

> Returns a list of organizations.
>
> *Example url:* `/common/organizations/`
> *Template file:* `common/templates/common/organization/organization_list.html`

class `hatecrimedb.common.views.OrganizationCreate`(***kwargs*)¶

> Create Organization view.
>
> *Example url:* `/common/organizations/create/`

Template file: `common/templates/common/organization/organization_form.html`

class `hatecrimedb.common.views.OrganizationDetail`(***kwargs*)¶

Organization Detail view.

Example url: `/common/organizations/1/`
Template file: `common/templates/common/organization/organization_detail.html`

class `hatecrimedb.common.views.OrganizationUpdate`(***kwargs*)¶

Organization update view.

Example url: `/common/organizations/1/update/`
Template file: `common/templates/common/organization/organization_form.html`

class `hatecrimedb.common.views.OrganizationDelete`(***kwargs*)¶

Organization deletion view.

Example url: `/common/organizations/1/delete/`
Template file: `templates/confirm_delete.html`

5.2.3. Reusable Views (Mixins)¶

The following mixins are used by various apps of the system.

class `hatecrimedb.common.viewmixins.AutocompleteBase`(***kwargs*)¶

Autocomplete base that produces.

`get_queryset()`¶

Filter the queryset with GET[‘q’].

class `hatecrimedb.common.viewmixins.DynamicPaginateByMixin`¶

Gets page size from url parameters.

class `hatecrimedb.common.viewmixins.EmptyInlineFormsetQuerySetMixin`¶

Mixin for CreateWithInlinesView and UpdateWithInlinesView that sets queryset to none() for selected inline models. Useful when you want to only add instances via inline formset.

class `hatecrimedb.common.viewmixins.FilterViewWithActiveFiltersMixin`¶

Adds “active_filters’ to a context with a list of active filters and their values. Shoudl be used along with FilterView class.

class `hatecrimedb.common.viewmixins.FormWithInlinesInvalidMessageMixin`¶

Mixin allows you to set static message which is displayed by Django’s messages framework through a static property on the class or programmatically by overloading the get_form_invalid_message method.

`forms_invalid`(*form*, *inlines*)¶

If the form or formsets are invalid, show error message.

class `hatecrimedb.common.viewmixins.FormWithInlinesValidMessageMixin`¶

Mixin allows you to set static message which is displayed by Django’s messages framework through a static property on the class or programmatically by overloading the get_form_valid_message method.

`forms_valid`(*form*, *inlines*)¶

If the form and formsets are valid, show the success message.

class `hatecrimedb.common.viewmixins.GenericDelete`(***kwargs*)¶

A generic deletion mixin that acts as a base for other Delete views of the system.

`get_context_data`(***kwargs*)¶

Insert the single object into the context dict.

`get_form_valid_message()`¶

Validate that form_valid_message is set and is either a unicode or str object.

class `hatecrimedb.common.viewmixins.HasHistoryStreamMixin`¶

Adds history stream context to the current object. Can be used to display object history stream and / or object history stream pagination.

class `hatecrimedb.common.viewmixins.HasPermissionMixin`¶

A mixin which verifies that the current user has all specified permissions.

class `hatecrimedb.common.viewmixins.UserFormsetKwargsMixin`¶

CBV mixin which puts the user from the request into the formset kwargs. Note: Using this mixin requires you to pop the *user* kwarg out of the dict in the super of your formset form’s *__init__*.

Works in conjunction with extra_views.advanced.InlineFormSet.

`get_extra_form_kwargs()`¶

Returns extra keyword arguments to pass to each form in the formset

class `hatecrimedb.common.viewmixins.ViewWithSortsSupportMixin`¶

Adds a dynamic, provided in url, sort support to a view.

5.3. hatecrimedb.incidents (incidents)¶

This app holds a set of models, views and templates that are responsible for creation, update, viewing and deletion of **Reports** (`Report`) and **Incidents** (`Incidents`). It also provides all necessary supporting models.

5.3.1. Main models¶

class `hatecrimedb.incidents.models.Report`(**args*, ***kwargs*)¶

Represents **Report** items.

Has foreign keys to `User` via `author` and `last_editor` fields.

Has many to many relationships with `Organization`, `Language`, `Country`.

class `hatecrimedb.incidents.models.Incident`(**args*, ***kwargs*)¶

Represents **Incident** items.

Has foreign keys to `Report`, `Incident`, and `User` via `author` and `last_editor` fields.

Has many to many relationships with `BiasMotivation`, `TypeOfCrime`, `Country`, `Organization`, `Crime`, `TypeOfCrime`, `BiasIndicator`, `Perpetrator`, `Property`.

class `hatecrimedb.incidents.models.MultipleIncidents`(**args*, ***kwargs*)¶

Represents **Multiple Incidents** *(aka Statistical Incident)* items.

Has foreign keys to `Report`, `Country`, and `User` via `author` and `last_editor` fields.

Has many to many relationships with `BiasMotivation`, `TypeOfCrime`.

5.3.2. Supporting models¶

class `hatecrimedb.incidents.models.ReportFile`(**args*, ***kwargs*)¶

Represents **Report file** items. Stores file attachments for `Report`.

Has foreign keys to `Report`.

class `hatecrimedb.incidents.models.ReportComment`(**args*, ***kwargs*)¶

Represents **Report comment** items. Stores comments for `Report`.

Has foreign keys to `Report`.

class `hatecrimedb.incidents.models.IncidentFile`(**args*, ***kwargs*)¶

Represents **Incident file** items. Stores file attachments for `Incident`.

Has foreign keys to `Incident`.

class `hatecrimedb.incidents.models.IncidentComment`(**args*, ***kwargs*)¶

Represents **Incident comment** items. Stores comments for `Incident`.

Has foreign keys to `Incident`.

class `hatecrimedb.incidents.models.Victim`(**args*, ***kwargs*)¶

Represents **Victim** items.

Has foreign keys to `Incident`, `Author`, `Last Editor`, `Injury`

Has many to many relationships with `GroupAffiliationTag`.

class `hatecrimedb.incidents.models.GroupAffiliationTag`(**args*, ***kwargs*)¶

Represents **Group Affiliation** items.

class `hatecrimedb.incidents.models.BiasMotivation`(**args*, ***kwargs*)¶

Represents **Bias motivation** items.

class `hatecrimedb.incidents.models.BiasIndicator`(**args*, ***kwargs*)¶

Represents **Bias indicator** items.

5.3.3. Database tables¶

- Every model has a corresponding database table.
- Every model has an index on the primary key which is `ID`.
- Every model has an index on every foreign key.

5.3.4. Views (Controllers), Routes and Templates¶

class `hatecrimedb.incidents.views.ReportList`(***kwargs*)¶

Page with report listing.

Example url: `/incidents/reports/`
Template file: `incidents/templates/incidents/report_list.html`

class `hatecrimedb.incidents.views.ReportDetail`(***kwargs*)¶

Individual report page.

Example url: `/incidents/reports/56/`
Template file: `incidents/templates/incidents/report_detail.html`

class `hatecrimedb.incidents.views.ReportCreate`(***kwargs*)¶

Create new report page.

Example url: `/incidents/reports/create/`
Template file: `incidents/templates/incidents/report_create.html`

class `hatecrimedb.incidents.views.ReportUpdate`(***kwargs*)¶

Update a report page.

Example url: `/incidents/reports/56/update/`
Template file: `incidents/templates/incidents/report_form.html`

class `hatecrimedb.incidents.views.ReportDelete`(***kwargs*)¶

Delete a report page.

Example url: `/incidents/reports/56/delete/`
Template file: `templates/confirm_delete.html`

class `hatecrimedb.incidents.views.IncidentList`(***kwargs*)¶

Page with incident listing.

Example url: `/incidents/incidents/`
Template file: `incidents/templates/incidents/incident_list.html`

class `hatecrimedb.incidents.views.IncidentDetail`(***kwargs*)¶

Individual incident page.

Example url: `/incidents/incidents/56/`
Template file: `incidents/templates/incidents/incidents_detail.html`

class `hatecrimedb.incidents.views.IncidentCreateForReport`(***kwargs*)¶

Create a new incident page for a report.

Example url: `/incidents/reports/81/create-incident/`
Template file: `incidents/templates/incidents/incident_form.html`

class `hatecrimedb.incidents.views.IncidentUpdate`(***kwargs*)¶

Update an incident page.

Example url: `/incidents/incidents/1675/update/`
Template file: `incidents/templates/incidents/incident_form.html`

class `hatecrimedb.incidents.views.IncidentDelete`(***kwargs*)¶

Delete an incident page.

Example url: `/incidents/incidents/1675/delete/`
Template file: `templates/confirm_delete.html`

5.3.5. Additional functionality¶

- `Report` and `Incident` have states that are managed via finite state machine approach. All states available can be found on classes in `STATE_CHOICES` property.
- `Report` and `Incident` field data changes are tracked via custom module `model_history`.

5.3.6. Report states¶

The editorial workflow of Report items is governed by their state. A report may be edited / viewed by particular group of users depending on its state. Every `Report` should have a single state value assigned to it. For example *Editing* or *Ready for publication*.

Below you will find a diagram of the state flow for the `Report`.

_images/report_states_top.png _images/report_states_bottom.png

A list of available states can be found in `Report.STATE_CHOICES` property.

5.3.7. Incident states¶

`Incident` also employ states, however these are optional for an `Incident`.

Here states are used to indicate additional status of the `Incident`, for example: *Duplicate* or *Needs review*.

Below you will find a diagram of the state flow for the `Incident`.

_images/incident_states.png

A list of available states can be found in `Incident.STATE_CHOICES` property.

5.4. hatecrimedb.crimes (crimes)¶

This app holds a set of models that are responsible for storage of crime-related information. It holds no special functionality besides providing models.

5.4.1. Main models¶

class `hatecrimedb.crimes.models.Crime`(**args*, ***kwargs*)¶

> Represents **Crime** items.
>
> Has foreign keys to `TypeOfCrime`.

class `hatecrimedb.crimes.models.TypeOfCrime`(**args*, ***kwargs*)¶

> Represents **Type of crime** items.

5.4.2. Supporting models¶

class `hatecrimedb.crimes.models.Perpetrator`(**args*, ***kwargs*)¶

> Represents **Perpetrator** items.

class `hatecrimedb.crimes.models.Injury`(**args*, ***kwargs*)¶

> Represents **Injury** items.

class `hatecrimedb.crimes.models.Property`(**args*, ***kwargs*)¶

> Represents **Property** items.

5.4.3. Database tables¶

- Every model has a corresponding database table.
- Every model has an index on the primary key which is `ID`.
- Every model has an index on every foreign key.

5.4.4. Views (Controllers), Routes and Templates¶

This app doesn’t have any views, routes, and templates.

5.5. hatecrimedb.model_history (model_history)¶

This app holds custom functionality that allows `Report` and `Incidents` to track and log field data changes.

For its diffing functionality, the app depends on a 3rd party package DeepDiff (https://github.com/seperman/deepdiff).

5.5.1. Work principle¶

When the supported model instance is saved, model_history compares the previous state of that instance with its updated state. Then, the model saves calculated differences in the database.

5.5.2. Models¶

This app doesn’t have any models.

5.5.3. Views (Controllers), Routes and Templates¶

This app doesn’t have any views, routes, and templates.

5.6. hatecrimedb.private_messages (private_messages)¶

This app provides a private messaging functionality. Users can send and receive private messages from other authenticated users.

5.6.1. Models¶

class `hatecrimedb.private_messages.models.Message`(**args*, ***kwargs*)¶

> Represents **Private message** items.
>
> Has foreign keys to `Thread`, `User`.

class `hatecrimedb.private_messages.models.Thread`(**args*, ***kwargs*)¶

Represents **Thread** items.

Has foreign key to `User`.

Has many to many relationships with `django.contrib.auth.models.Group`, `User`.

class `hatecrimedb.private_messages.models.UserThread`(**args*, ***kwargs*)¶

Represents **User thread** items.

Has foreign keys to `Thread`, `User`.

5.6.2. Database tables¶

- Every model has a corresponding database table.
- Every model has an index on the primary key which is `ID`.
- Every model has an index on every foreign key.

5.6.3. Views (Controllers), Routes and Templates¶

class `hatecrimedb.private_messages.views.InboxView`(***kwargs*)¶

View inbox thread list.

Example url: `/messages/`
Template file: `private_messages/templates/private_messages/thread_list.html`

class `hatecrimedb.private_messages.views.ThreadView`(***kwargs*)¶

View a single Thread or POST a reply.

Example url: `/messages/thread/1/`
Template file: `private_messages/templates/private_messages/thread_detail.html`

class `hatecrimedb.private_messages.views.MessageCreateView`(***kwargs*)¶

Create a new thread message.

Example url: `/messages/create/`
Template file: `private_messages/templates/private_messages/message_create.html`

class `hatecrimedb.private_messages.views.ThreadDeleteView`(***kwargs*)¶

Delete a thread.

Example url: `/messages/thread/1/delete/`
Template file: `private_messages/templates/private_messages/thread_confirm_delete.html`

5.7. hatecrimedb.users (users)¶

This app is responsible for handling **User** (`User`) accounts functionality. It includes user profiles, user registration, password restoration and user dashboards.

The primary model `User` extends core Django functionality with additional profile related fields.

5.7.1. Models¶

class `hatecrimedb.users.models.User`(**args*, ***kwargs*)¶

Implements a fully featured User model with admin-compliant permissions.

Has encrypted `password` field that securely stores user passwords in database.

5.7.2. Database tables¶

- All users stores in a corresponding `users` database table.
- User model has index on the primary key which is `ID`.
- Passwords stored as encrypted.

5.7.3. Views (Controllers), Routes and Templates¶

class `hatecrimedb.users.views.UserDashboardView`(***kwargs*)¶

A dashboard view which acts as the main entrypoint for a user.

Example url: `/users/1/dashboard/`
Template file: `users/templates/users/user_dashboard.html`

class `hatecrimedb.users.views.UserList`(***kwargs*)¶

A view listing all users.

Example url: `/users/`
Template file: `users/templates/users/user_list.html`

class `hatecrimedb.users.views.UserDetailView`(***kwargs*)¶

A view displaying user profile.

Example url: `/users/1/`
Template file: `users/templates/users/user_detail.html`

class `hatecrimedb.users.views.UserCreate`(***kwargs*)¶

Create a user view.

Example url: `/users/create/`
Template file: `users/templates/users/user_create.html`

class `hatecrimedb.users.views.UserUpdate`(***kwargs*)¶

A view for user settings.

Example url: `/users/1/update/`
Template file: `users/templates/users/user_form.html`

class `hatecrimedb.users.views.UserDelete`(***kwargs*)¶

Delete a user view.

Example url: `/users/1/delete/`
Template file: `templates/confirm_delete.html`

class `hatecrimedb.users.views.reset_password`¶

A view for password reset.

Example url: `/users/1/password-reset/`
Template file: `users/templates/users/_password_form.html`

5.7.4. User roles¶

There are several user roles, each having a different set of permissions: * Administrator * Coordinator * Editor * Analyst * Reviewer

A list of per-group permissions can be found and edited in the admin panel here: `/backdoor/auth/group/`.

5.8. hatecrimedb.dbbackup_email (dbbackup_email)¶

The app is an addon to 3rd party Django module **Django Database Backup** (https://django-dbbackup.readthedocs.io/en/stable/). It uses capabilities of **Django Database Backup** to make a backup of the database and media files.

It provides a `backup_and_email` management command, that should be scheduled as a cron job.

When the command is called, it makes a backup of database and media files, stores them locally and sends out download link via email to site administrators.

A list of email addresses of site administrators is defined in `config.settings.production.DBBACKUP_EMAIL_SEND_TO` variable.

To send email notifications about backups, this module requires a mail server to be set and available.

5.8.1. Command usage¶

You can run the command as simple as:

```
python manage.py backup_and_email
```

5.8.2. Models¶

This module has no models.

5.8.3. Views (Controllers) and Routes¶

class `hatecrimedb.dbbackup_email.views.download_file`¶

A view that returns a backup file download.

Backup files are stored in publicly inaccessible directory. This view checks if a user has access to the given backup file and if yes, it sends the file from the backup storage for download.

Example url: `/backups-storage/download/database-hatecrime-prod-2017-11-07-00000.psql.gz/`

5.8.4. Templates¶

The backup email message is produced by the following templates:

- `dbbackup_email/templates/dbbackup_email/backup_email_subject.txt` for subject line,
- `dbbackup_email/templates/dbbackup_email/backup_email_body.html` for body text.

These can be used to tweak the subject and body of the email.

6. Information about forks of 3rd party apps¶

To meet project needs, some 3rd party have been forked and customized.

Below you can find a list of such apps with a description of their functionality.

6.1. bootstrapform¶

This app provides a simple Django template tag to work with Bootstrap (http://getboostrap.com) theme.

At the moment of the development, this app looked abandoned, so we've forked it to make it work with Django 1.11. Original repo url is https://github.com/tzangms/django-bootstrap-form.

6.2. djangoformsetjs¶

This app provides a wrapper for a javascript formset wrapper. It allows formsets to be added dynamically.

We use formsets in file attachments in `Report` and `Incident`.

The original app has been abandoned, so we've forked it to make it work with the latest version of Django. Original repo url is https://bitbucket.org/tim_heap/django-formset-js.

6.3. bootstrapform¶

The app provides functionality of remembering selected files in form on validation errors, so users don't lose already attached files when form returns to them with a validation error.

At the moment of the development, this app looked abandoned, so we've forked it to make it work with Django 1.11. Original repo url is https://github.com/un1t/django-file-resubmit.

6.4. flexible_date¶

The app provides a new model field that allows to save partial dates (i.e. Year only, year and month only).

The field is used by `Incident` model.

It is a fork of the project that has been almost totally rewritten to fit our needs.

7. System Requirements¶

There are following system requirements for the system:

7.1. Server specs¶

- Operation system: Linux (Ubuntu 16.04 LTS or CentOS 7);
- CPU: at least 2 cores;
- RAM: at least 2GB.

7.2. Service specs¶

- Execution environment: Python 3.6 (https://www.python.org/);
- HTTP Server: Apache 2.2 or 2.4 (https://httpd.apache.org/) or Nginx 1.12+ (https://nginx.org/en/);
- Apache Module for uWSGI support(https://uwsgi-docs.readthedocs.io/en/latest/Apache.html) if Apache server is in use;
- Database: PostgreSQL 9.6 (https://www.postgresql.org/).

7.3. Python specific requirements¶

- Python Server: Uwsgi (https://uwsgi-docs.readthedocs.io/en/latest/);
- PIP tool for package installation: https://pip.pypa.io/en/stable/;
- Virtualenv to create isolated Python environments: https://virtualenv.pypa.io/en/stable/.

7.4. Other requirements¶

- The system runs behind HTTPS, so active SSL certificate should be installed and configured on the server.

8. Installation¶

1. Create a directory for the project on the server:

   ```
   root@server# mkdir -p /var/www/vhosts/hatecrimedb.odihr.pl/
   ```

2. Change the current directory to the project dir:

   ```
   root@server# cd /var/www/vhosts/hatecrimedb.odihr.pl/
   ```

3. Create a virtual environment for the project:

   ```
   root@server# virtualenv .
   ```

4. Activate virtual environment:

   ```
   root@server# source ./bin/activate
   ```

5. Checkout code base to /var/www/vhosts/hatecrimedb.odihr.pl/app/:

   ```
   (hatecrime) root@server# git clone git@git.assembla.com:waat/hatecrime.2.git app
   ```

6. The app source code should be available at:

   ```
   (hatecrime) root@server# /var/www/vhosts/hatecrimedb.odihr.pl/app/
   ```

7. Copy config file to the config dir:

```
(hatecrime) root@server# cp /var/www/vhosts/hatecrimedb.odihr.pl/app/.env.example /var/www/vhosts/hatecrimedb.odihr.pl/app/code/.env
```

8. Edit .env file and set blank values
9. Install dependencies. While in `/var/www/vhosts/hatecrimedb.odihr.pl/app` run the following command:

```
(hatecrime) root@server# pip install -r requirements.txt
```

10. Create a config file for uwsgi that follows the following template:

```
[uwsgi]
uid = www-data
gid = www-data
virtualenv = /var/www/vhosts/hatecrimedb.odihr.pl/
chdir = /var/www/vhosts/hatecrimedb.odihr.pl/app/code
uwsgi-socket = 127.0.0.1:3032
wsgi-file = /var/www/vhosts/hatecrimedb.odihr.pl/app/code/config/wsgi.py
processes = 3
master = True
touch-reload=/var/www/vhosts/hatecrimedb.odihr.pl/app/code/config/wsgi.py
```

11. Save the file as `/var/www/vhosts/hatecrimedb.odihr.pl/uwsgi.ini`
12. Restore the database from a backup.
13. Run migrations and collectatic while in project's directory:

```
(hatecrime) root@server# cd /var/www/vhosts/hatecrimedb.odihr.pl/app/code
(hatecrime) root@server# python manage.py collectstatic
(hatecrime) root@server# python manage.py migrate
```

14. Run WSGI server via the command:

```
(hatecrime) root@server# /var/www/vhosts/hatecrimedb.odihr.pl/bin/uwsgi --ini /var/www/vhosts/hatecrimedb.odihr.pl/app/uwsgi.ini
```

15. Set the web server configuration to point to the WSGI domain, below you will find an example of the vhost record for Apache 2 (current setup):

```
<VirtualHost 46.41.131.169:80>

  ServerName hatecrimedb.odihr.pl
  ServerAlias www.hatecrimedb.odihr.pl
  ProxyPass / uwsgi://127.0.0.1:3032/

  <Location /static/>
    ProxyPass !
  </Location>

  <Location /media/>
    ProxyPass !
  </Location>

  <Location "/media">
      SetHandler None
  </Location>
  <LocationMatch "\.(jpg|gif|png|js|css)$">
      SetHandler None
  </LocationMatch>

  Alias /media/ "/var/www/vhosts/hatecrimedb.odihr.pl/app/code/hatecrimedb/media/"
  Alias /static/ "/var/www/vhosts/hatecrimedb.odihr.pl/app/code/staticfiles/"
  ErrorLog /var/www/vhosts/hatecrimedb.odihr.pl/logs/error.log
  LogLevel warn
  CustomLog /var/www/vhosts/hatecrimedb.odihr.pl/logs/access.log combined

  RewriteEngine on
  RewriteCond %{SERVER_NAME} =www.hatecrime.odihr.pl [OR]
  RewriteCond %{SERVER_NAME} =hatecrimedb.odihr.pl
  RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]

</VirtualHost>
```

16. Restart apache.

9. Deployment¶

9.1. Manual deployment¶

To deploy a new code change do the following:

1. Change directory into project's root:

```
root@server# cd /var/www/vhosts/hatecrimedb.odihr.pl/app/
```

2. Pull changes from git:

```
root@server# git pull origin master
```

3. Activate virtual environment:

```
root@server# source ./bin/activate
```

4. Change into code directory:

```
(hatecrime) root@server# cd /code/
```

5. Install dependencies from requirements.txt:

```
(hatecrime) root@server# pip install -r requirements.txt
```

6. Run collectstatic and migration:

```
(hatecrime) root@server# python manage.py collectstatic
python manage.py migrate
```

7. Touch wsgi.py file to force WSGI server restart:

```
root@server# touch config/wsgi.py
```

9.2. Automated deployment¶

There's also a possibility to do automated deployments via Fabric (http://www.fabfile.org/) and Ansible (https://www.ansible.com/) devops tools.

Requirements:

On your local dev machine you need to have:

- Fabric installed (http://www.fabfile.org/),
- Ansible installed (https://www.ansible.com/).

Inside a `deploy` directory of the root of the project you can find Ansible playbooks that orchestrate the deployment process.

In order to deploy a code change to the production server you just need to run the following command:

```
root@server# fab live deploy
```

The system will ask for the master password and start the deployment process automatically.

10. Maintenance¶

Security updates of all underlying system components should be applied promptly by the system owner:

This includes but not limits to:

- Operation system
- Web server
- Database
- Execution environment

10.1. Django framework updates¶

We recommend following Django security updates for the Django 1.11.

Django 1.11 is the LTS version, which means it will get security updates until at least April 2020. More info on supported versions can be found here: https://www.djangoproject.com/download/#supported-versions.

Applying security update for the Django core is as simple as running the command:

```
(hatecrime) root@server# pip install "Django>=1.11,<1.12" --upgrade
```

Note

The decision, whether to upgrade to Django 2.x, depends solely on the system owner and may result in the system not working correctly, so the update should be done with caution.

10.2. 3rd party modules security updates¶

3rd party modules used in the system may also issue security updates, which we also recommend applying on a timely basis.

Note

The system should be checked if it runs correctly after every update of every component.

11. HatecrimeDB Public API Integration module for Drupal¶

The system exposes public REST API via hatecrimedb.api (api) module.

Such API can be consumed by 3rd party clients.

Current implementation of such client is a Drupal module **hcrw_api_client** that's the part of the Drupal installation.

The module itself integrates Drupal with a Vue v2.3 (https://vuejs.org/) based JavaScript application that consumes the api.

11.1. Source code¶

The source code of the module can be found in Drupal installation under `sites/all/modules/custom/hcrw_api_client` directory.

The source code of the Vue app is located inside `app` directory of the module.

11.2. Main components¶

The app provides several components that are used to display HatecrimeDB data on http://hatecrime.osce.org/ website:

1. `IncidentsSearch.vue` component that's responsible for display of the *http://hatecrime.osce.org/incidents* page.
2. `BiasMotivationsGraph.vue` component displays **Incidents reported by civil** graph on the country pages. Example: *http://hatecrime.osce.org/belgium*.

3. `IncidentsDownloadDataButton.vue` component displays **Download incident data** button on the country pages. Example: *http://hatecrime.osce.org/belgium*.
4. `BiasMotivationItem.vue` component displays table with incidents related to a single bias motivation. Example: *http://hatecrime.osce.org/belgium*.
5. `BiasMotivationCountries.vue` component displays **Incidents were reported on these States** panel on bias motivation pages. Example: *http://hatecrime.osce.org/what-hate-crime/bias-against-muslims*.
6. `BiasMotivationSummaryTable.vue` component displays **Overview of incidents reported by other sources** summary table on bias motivation pages. Example: *http://hatecrime.osce.org/what-hate-crime/bias-against-muslims*.
7. `BiasMotivationIncidentsTable.vue` component displays a list of incidents related to the bias motivation. Example: *http://hatecrime.osce.org/what-hate-crime/bias-against-muslims*.

11.3. Code updates¶

To make code updates, a Node.JS v8.9 environment (https://nodejs.org) should be installed on a developer's machine. Please note, that Node is not required to be installed on the production server.

1. Install all necessary dependencies. When in `sites/all/modules/custom/hcrw_api_client/app`, run:

   ```
   npm install
   ```

2. To start development, run:

   ```
   npm start
   ```

3. While in development, to see updates in Drupal make sure that `HCRW_API_DEBUG` in Drupal is set to `FALSE`. In order to do that, edit file `sites/all/modules/custom/hcrw_api_client/hcrw_api_client.module` on line **7**:

   ```
   define('HCRW_API_DEBUG', FALSE);
   ```

4. When the development is finished and the app is ready to be deployed, create a production build, run:

   ```
   npm run build
   ```

5. And switch `HCRW_API_DEBUG` value back to `TRUE`, so Drupal will know that it should use the production build:

   ```
   define('HCRW_API_DEBUG', TRUE);
   ```

12. References¶

1. Django, a web framework (https://www.djangoproject.com/),
2. Django documentation (https://docs.djangoproject.com/en/1.11/),
3. Python, a programming language (https://www.python.org/),
4. uWSGI, a python server (https://uwsgi-docs.readthedocs.io/en/latest/),
5. Virtualenv, a tool to create isolated Python environments (https://virtualenv.pypa.io/en/stable/),
6. PIP, Python package manager (https://pip.pypa.io/en/stable/),
7. Apache, a web server (https://httpd.apache.org/),
8. Nginx, a web server (https://nginx.org/en/),
9. Postgres, a database server (https://nginx.org/en/),
10. VueJS, a frontend framework (https://vuejs.org/),
11. Ansible, a deployment tool (https://www.ansible.com/),
12. Fabric, a deployment tool (http://fabfile.org).
