gino-admin

Release 0.2.3

Dec 29, 2021

Contents

CHAPTER 1

Installation

First of all you need to install Gino-Admin to your project environment

```
$ pip install gino-admin==0.2.4
```

CHAPTER 2

How to use

You can find several code examples in ‘examples’ folder - https://github.com/xnuinside/gino-admin/tree/master/examples .

2.1 How to run Gino-Admin

CHAPTER 3

Run with Cli

```
gino-admin run #module_name_with_models -d postgresql://%(DB_USER):%(DB_PASSWORD)@
↪%(DB_HOST):%(DB_PORT)/%(DB)

gino-admin run --help # use to get cli help
Optional params:
    -d --db
        Expected format: postgresql://%(DB_USER):%(DB_PASSWORD)@%(DB_HOST):%(DB_PORT)/
↪%(DB)
        Example: postgresql://gino:gino@%gino:5432/gino (based on DB settings in␣
↪examples/)
        Notice: DB credentials can be set up as  env variables with 'SANIC_' prefix
    -h --host
    -p --port
    -c --config Example:  -c "presets_folder=examples/base_example/src/csv_to_upload;
↪some_property=1"
                Notice: all fields that not supported in config will be ignored, like
↪'some_property' in example
    --no-auth  Run Admin Panel without Auth in UI
    -u --user Admin User login & password
        Expected format: login:password
        Example: admin:1234
        Notice: user also can be defined from env variable with 'SANIC_' prefix -␣
↪check Auth section example
```

Example:

```
gino-admin run examples/run_from_cli/src/db.py --db postgresql://
↪gino:gino@localhost:5432/gino -u admin:1234
```

CHAPTER 4

Run Admin Panel as Standalone App (no matter that framework you use in main app)

You can use Gino Admin as stand alone web app. Does not matter what Framework used for your main App and that Gino Ext is used to init Gino().

Code example in: examples/fastapi_as_main_app How to run example in: examples/fastapi_as_main_app/how_to_run_example.txt

You need to create **admin.py** (for example, you can use any name) to run admin panel:

```
import os

from gino_admin import create_admin_app
# import module with your models
import models

# gino admin uses Sanic as a framework, so you can define most params as environment␣
↪variables with 'SANIC_' prefix
# in example used this way to define DB credentials & login-password to admin panel

# but you can use 'db_uri' in config to define creds for Database
# check examples/colored_ui/src/app.py as example

os.environ["SANIC_DB_HOST"] = os.getenv("DB_HOST", "localhost")
os.environ["SANIC_DB_DATABASE"] = "gino"
os.environ["SANIC_DB_USER"] = "gino"
os.environ["SANIC_DB_PASSWORD"] = "gino"


os.environ["SANIC_ADMIN_USER"] = "admin"
os.environ["SANIC_ADMIN_PASSWORD"] = "1234"

current_path = os.path.dirname(os.path.abspath(__file__))
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    # host & port - will be used to up on them admin app
    # config - Gino Admin configuration - check docs to see all possible properties,
    # that allow set path to presets folder or custom_hash_method, optional parameter
    # db_models - list of db.Models classes (tables) that you want to see in Admin␣
↪Panel
    create_admin_app(
        host="0.0.0.0",
        port=os.getenv("PORT", 5000),
        db=models.db,
        db_models=[models.User, models.City, models.GiftCard, models.Country],
        config={
            "presets_folder": os.path.join(current_path, "csv_to_upload")},
    )
```

All environment variables you can move to define in docker or .env files as you wish, they not needed to be define in '.py', this is just for example shortness.

CHAPTER 5

Add Admin Panel to existed Sanic application as '/admin' route

Create in your project 'admin.py' file and use `add_admin_panel` from from gino_admin import add_admin_panel

Code example in: examples/base_example How to run example in: examples/base_example/how_to_run_example.txt

Example:

```
from from gino_admin import add_admin_panel


# your app code


add_admin_panel(
    app, db, [User, Place, City, GiftCard], custom_hash_method=custom_hash_method
)
```

Where:

- 'app': your Sanic application
- 'db' : from gino.ext.sanic import Gino; db = Gino() and
- [User, Place, City, GiftCard] - list of models that you want to add in Admin Panel to maintain
- custom_hash_method - optional parameter to define you own hash method to encrypt all '_hash' columns of your Models.

In admin panel _hash fields will be displayed without '_hash' prefix and fields values will be hidden like '**'

CHAPTER 6

Cli

Run Gino Admin Panel with Cli

```
gino-admin run #module_name_with_models -d postgresql://%(DB_USER):%(DB_PASSWORD)@
↪%(DB_HOST):%(DB_PORT)/%(DB)

gino-admin run --help # use to get cli help
Optional params:
    -d --db
        Expected format: postgresql://%(DB_USER):%(DB_PASSWORD)@%(DB_HOST):%(DB_PORT)/
↪%(DB)
        Example: postgresql://gino:gino@%gino:5432/gino (based on DB settings in␣
↪examples/)
        Notice: DB credentials can be set up as  env variables with 'SANIC_' prefix
    -h --host
    -p --port
    -c --config Example:  -c "presets_folder=examples/base_example/src/csv_to_upload;
↪some_property=1"
                Notice: all fields that not supported in config will be ignored, like
↪'some_property' in example
    --no-auth  Run Admin Panel without Auth in UI
    -u --user Admin User login & password
        Expected format: login:password
        Example: admin:1234
        Notice: user also can be defined from env variable with 'SANIC_' prefix -␣
↪check Auth section example
```

Example:

```
gino-admin run examples/run_from_cli/src/db.py --db postgresql://
↪gino:gino@localhost:5432/gino -u admin:1234
```

CHAPTER 7

Examples

*How to use Composite CSV *Sample Example with Presets *Use GinoAdmin with FastAPI Example

CHAPTER 8

Configure Gino Admin

You can define in config:

- presets_folder: path where stored predefined DB presets
- custom_hash_method: method that used to hash passwords and other data, that stored as '_hash' columns in DB, by default used pbkdf2_sha256.encrypt
- composite_csv_settings: describe some rules how to parse and load Composite CSV files
- name: project name, that will be displayed in UI. By default it shows: "Sanic-Gino Admin Panel"
- csv_update_existed: By default 'csv_update_existed = True'. This mean if you upload CSV with rows with unique keys, that already exist in DB - it will update all fields with values from CSV. You can turn off it with set 'csv_update_existed = False'.
- route: Route where will be served (that will be used to access) Admin panel. By default, used '/admin' route
- round_number: How much symbols display in floats in UI (default 3)
- db_uri: pass path & credentials to DB with db_uri, example: "postgresql://local:local@localhost:5432/gino_admin"
- ui: customize ui with config settings

8.1 ui

colors

In UI section exist property 'colors' where you can set up colors that will be used for:

- Primary buttons. Property: buttons
- Second buttons. Property: buttons_second
- Alert buttons (actions that something remove/reset - deleted, drop db and etc). Property: buttons_alert
- Tables headers. Property: table

- Alert Tables (as in Init DB). Property: table_alert
- Footer background. Property: footer
- Header background. Property: header

How to pass. Example: config = {

“ui” [{] “colors”: {

“buttons”: “orange”, “buttons_alert”: “pink” }

}

}

8.2 composite_csv_settings

composite_csv_settings allow to define multiple tables as one alias

For example, in our example project with composite CSV we have 3 huge different categories separated by tables (they have some different columns) - Camps, Education(courses, lessons, colleges and etc.) and Places(Shopping, Restaurants and etc.) But we want to avoid duplicate similar columns 3 times, so we can call those 3 tables by one alias name, for example: ‘area’ and some column to understand that exactly this is an ‘area’ - capms, educations or places table for this we need to define ‘type_column’ we don’t use in any model column ‘type’ so we will use this name for type-column

So, now let’s define **composite_csv_settings**

```
composite_csv_settings={
    "area": {"models": (Place, Education, Camp), "type_column": "type"}
}
```

This mean, when we see in CSV-header ‘area’ this is data for one of this 3 models, to identify which of this 3 models - check column with header ‘area:type’. In type column values must be same 1-to-1 as table names.

Check source code with example: examples/composite_csv_example

And table sample for it: https://docs.google.com/spreadsheets/d/1ur63acwWExyjWouZ1WEkUxCX73vOcdXzCrEYc7cPhTg/edit?usp=sharing

You also can define table name as ‘pattern’:

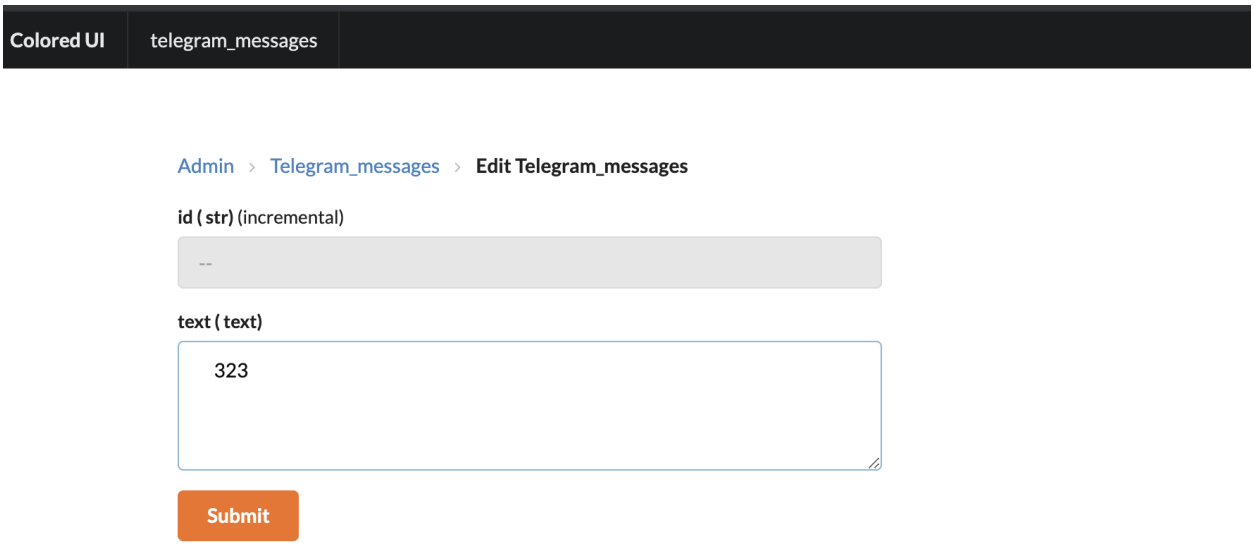
```
composite_csv_settings={
    "area": {"models": (SomeModel, SomeModel2, SomeModel3), "pattern": "*_postfix"}
}
```

This mean - to understand that this is a DB - take previous table from CSV in row and add ‘_postfix’ at the end.

CHAPTER 9

UI Customization

9.1 Change UI Colors



Config object has section ‘ui’. In UI section now exist ‘colors’ where you can set up colors that will be used for:

- Primary buttons. Property: buttons
- Second buttons. Property: buttons_second
- Alert buttons (actions that something remove/reset - deleted, drop db and etc). Property: buttons_alert
- Tables headers. Property: table
- Tables with Alert headers (like in Init DB). Property: table_alerts

- Footer background. Property: footer
- Header background. Property: header

Admin panel used Semantic UI as CSS Framework so all names of possible colors is described and showed here: https://semantic-ui.com/usage/theming.html

(red: #B03060; orange #FE9A76; yellow: #FFD700; olive: #32CD32 green: #016936; teal : #008080; blue : #0E6EB8; violet: #EE82EE; purple: #B413EC; pink: #FF1493; brown: #A52A2A; grey : #A0A0A0; black: #000000;)

To change colors pass config as:

Example here: examples/colored_ui/

9.2 Set Custom Header

To do this just set provide config argument:

- name: project name, that will be displayed in UI. By default it shows: "Sanic-Gino Admin Panel"

Example:

```
if __name__ == "__main__":
create_admin_app(
    db.db,
    create_models_list(db),
    port=os.environ.get("PORT", "5000"),
    config={
        "name": "Colored UI"
        }
```

Example in github: https://github.com/xnuinside/gino-admin/blob/master/examples/colored_ui/src/app.py#L28

CHAPTER 10

Features

10.1 Supported Features

- Auth by login/pass with cookie check; Disable auth
- Multiple Users & Manage Admin users from Panel
- Create(Add new) item by one for the Model
- Search/sort in tables
- Upload/export data from/to CSV
- Delete all rows/per element
- Copy existed element (data table row)
- Edit existed data (table row)
- SQL-Runner (execute SQL-queries)
- Presets: Define order and Load to DB bunch of CSV-files
- Init DB (Full clean up behavior: Drop tables & Recreate)
- Deepcopy element (recursive copy all rows/objects that depend on chosen as ForeignKey)
- Composite CSV: Load multiple relative tables in one CSV-file
- Customize UI (switch colors, add custome title in header)
- History tracking of actions in DB (logs in admin panel)

10.2 Features in TODO List

- Add possible to add new Presets from GUI
- Select multiple for delete/copy

- Edit multiple items (?)
- Roles
- Filters in Table's columns

If you want to have any other feature - just open the github issue with description - https://github.com/xnuinside/gino-admin/issues

CHAPTER 11

API Interface

11.1 admin/api/auth

1.1 POST: admin/api/auth

Auth required to use API endpoints

To get auth JWT token:

In request header Authorization provide Basic b64decode login:password stroke

for user admin:1234 headers={"Authorization":"Basic YWRtaW46MTIzNA=="}

Not recommended: For fast and easy POC development also allowed provide plain text login:password stroke in Authorization header headers={"Authorization":"admin:1234"}

Response:

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJ1c2VyX2lkIjoxLCJleHAiOjE1ODkzNzI1MzZ9.IJZG9DV8ZCna7pjK7osUn9veI0Gc47d0Ts5pyGvu6JE
↪"
}
```

11.2 admin/api/presets

1.2 POST: admin/api/presets

protected: True (need to provide JWT token in Authorization header) Content-type: application/json

Request body:

- preset: must contain path to preset '.yml' file
- drop: flag to Drop & recreate tables in DB before upload preset (optional)

Purposes: easy call from tests env when need to drop/create DB from some tests datasets

11.3 admin/api/init_db

1.3 POST: admin/api/init_db

protected: True (need to provide JWT token in Authorization header) Empty request without body. Purposes: Clean up & recreate tables

CHAPTER 12

Presets

Presets allow you to load multiple CSV to DB in order by one click.

For this you need to define folder with DB presets. Inside folder you puy config files for Presets described in yml format. In config file you define order in what to load CSV-s files, that files used to populate tables.

To see more clear example check & run examples with CSV in examples/base_example folder.

Let's take a look on preset sample:

```
id: first_preset
name: First Preset
description: "Init DB with minimal data"
files:
  users: csv/user.csv
  gifts: csv/gift.csv
```

This mean preset contains 2 csv files, firstly will be loaded csv/user.csv to users table, second will be loaded csv/gift.csv to gifts table.

In order defined in yml, Gino-Admin will load csv files to models. 'files:' describe that file (right sight) must be loaded to the model (left side).

Don't forget to setup path to folder with presets like with **'presets_folder'** argument, by default it tries to find presets in 'presets/' folder.

```
...

current_path = os.path.dirname(os.path.abspath(__file__))

add_admin_panel(
    app,
    db,
    [User, Place, City, GiftCard, Country],
    custom_hash_method=custom_hash_method,
    presets_folder=os.path.join(current_path, "csv_to_upload"),
)
```

Check code samples in examples/base_example/

CHAPTER 13

Upload CSV Files

Exist 2 supported formats of uploaded CSV files:

- Upload table per model: one csv file contains data only for one table
- Composite CSV files: when you define several relative tables in one file

13.1 Composite CSV to Upload

Default upload from CSV allows to load CSV with data per table.

Composite CSV files allow to load data for several tables from one CSV files and don't define ForeignKey columns. You can define table from left to right and if previous table contain ForeignKey for the next table when as linked row will be taken value from current or previous row. This allow you to define one time Country and 10 cities for it. If it sounds tricky - check example DB schema and XLS example on google docs.

This useful if you want to fill DB with related data, for example, User has some GiftCards (ForeignKey - user.id), GiftCard can be spend to pay off for some Order (ForeignKey - gift_card.id). So you have set of data that knit together. If you works on some Demo or POC presentation - it's important to keep data consistent, so you want to define 'beautiful data', it's hard if you have 3-4-5 models to define in separate csv.

Composite CSV allow use CSV files with headers with pattern "table_name:column" and also allow to add aliases patterns

Check 'examples/composite_csv_example' code to check DB structure.

And XLS-table sample in Google Sheets: table example

countries:code	countries:count	cities:id	addresses:addr	area:type	area:title	area:theme	area:age	area:specialisat	area:category
RU	Russia	Moscow	Some street 1-2	educations	Private Kindergarden			Childrens	
			Another street 1-	camps	Best camp	Music	14-17		
			One more street	places	Joe's music cafe				Family's Cafes
			Park zone str 1-2	places	Russian Museum				Expositions
			Long Park zone s	places	Hermitage				Expositions
US	USA	New York	New street 1-2	camps	Zoo camp	Animals	6-8		
			New one more st	educations	Yoga center			Lifestyle	This field will be ig

Click - Download -> CSV and you will get result, that can be found in **examples/composite_csv_example/src/csv_to_upload**

Composite CSV can be loaded manual from any Model's Page where exist button 'Upload CSV' - it does not matter from that model you load.

Or you can define preset with Composite CSV and load it as preset. To use composite CSV you need to define key, that started with 'composite' word.

Example:

```
name: Composite CSV Preset
description: "Init DB with data from composite CSV"
files:
  composite_csv: csv/preset_a/users.csv
```

'composite_csv: csv/preset_a/users.csv' can be 'composite_any_key: csv/preset_a/users.csv'

You can use multiple composite CSV in one preset.

CHAPTER 14

Changelog

Actual changelog alway available here: https://github.com/xnuinside/gino-admin/blob/master/CHANGELOG.txt

14.1 Version 0.2.3 (current master)

1. Fix for the issue https://github.com/xnuinside/gino-admin/issues/35
2. 'gino' support tables removed from menu
3. Incremental Ids (fields) supported now in UI - they showed as disable inputs in Adding and Edit forms and not cause issue anymore.
4. Checkboxes are fixed and now work correct in the "False" state.

14.2 Version 0.2.2

1. Added support for types: JSONB, JSON and Time. Examples added as part of base_example - /Users/iuliia_volkova2/work/gino-admin/examples/base_example

2. **Main update**: Not needed to use *gino.ext.sanic* in models as it was previos.

Now you can use any ext if you need (for Fast Api for example) or pure Gino() and you not need to add to your models, any 'ifs' to have additional gino.ext.sanic to get possible work with gino admin.

All examples was changed according to the update.

4. Sanic was updated to 20.* version. Switched to use 'request.ctx.' in code
5. Minor things: all date, datetime and timepickers now have default value == to current time/date/datetime.
6. Tests: was updated structure of integraion tests

14.3 Version 0.2.1

1. Fixes:

1.1 Dependencies - removed unnecessary packages and added one lost for cli. Cli now works correct. 1.2 Login form now provide errors if you enter wrong user or passoword 1.3 Wrong attepts to login in Admin panel are adding to History now

2. Added possibility to customize UI colors with config.

Default colors schema also changed:

![Table view](docs/img/new_colors.png)

Config object now has section 'ui'. In UI section now exist 'colors' where you can set up colors that will be used for:

- Primary buttons. Property: buttons
- Second buttons. Property: buttons_second
- Alert buttons (actions that something remove/reset - deleted, drop db and etc). Property: buttons_alert
- Tables headers. Property: table
- Tables with Alert headers (like in Init DB). Property: table_alerts
- Footer background. Property: footer
- Header background. Property: header

Admin panel used SemanticUI as CSS Framework so all names of possible colors is described and showed here: https://semantic-ui.com/usage/theming.html

(red: #B03060; orange #FE9A76; yellow: #FFD700; olive: #32CD32 green: #016936; teal : #008080; blue : #0E6EB8; violet: #EE82EE; purple: #B413EC; pink: #FF1493; brown: #A52A2A; grey : #A0A0A0; black: #000000;)

To change colors pass config as:

```python
create_admin_app(
        host="0.0.0.0", port=os.getenv("PORT", 5000), db=example.models.db, db_models=db_models,
        config={
            "ui" [{] "colors": {"buttons": "orange", "buttons_alert": "pink"} },
            "db_uri": "postgresql://gino:gino@localhost:5432/gino"
        },
    )
```

Example here: examples/colored_ui/

3. Added example how to add all models from file with one method (to avoid import each model separate) - palced in *examples/colored_ui/src/app.py* method - **create_models_list**

4. Added valid input for Text columns as Text Area ![Text Area Inouts](docs/img/text_area.png)

14.4 Version 0.2.0:

1. **UI fixes**: - Data Picker was fixed, required fields now dispalayed with '* required' in UI. - Menu in header became scrollable, now you can see 20+ models without pain - Tables became scrollable horisontal - you can keep dozen columns and see them (hooray!) - in Add/edit forms now displayd the field type

2. **Major changes**: - **Limitation to have 'unique' rows was removed**. Now you not need any unique keys to make possible work with table in Admin panel. Just keep in mind that if you edit row - you will also edit all full 'dublicated' rows. So we try identify row by all fields. But if you have several full duplicates in rows - edit action will edit all of them.

Limits:

Deepcopy does not available for tables without primary keys right now.

- **Primary keys** now also used to identify unique rows. Now Admin Panel don't expect only 'unique' key in model. Now it firstly works with Primary Keys and only if primary key not exist in model - use 'unique' fields to identify unique rows. Also it supports Composite Primary keys (2 and more fields) in all type of operations: delete/update/insert/deepcopy/copy.
- **Schemas support**

Now if you work using the custom "schema" name - it's okay and supported by Admin Panel.

3. **Fixed in types support**:

- passing data as a string - now supported both Date & DateTime format (before correct work only DataTime format)
- parsing lists (for fields with ARRAY type), also parsed type inside array

4. **Types support improvement**:

- Added support for ARRAYS, TEXT, SmallInt, CHAR, Time

5. **New features**: - Added Users to Admin Panel - now you can add multiple users for the panel to track history of changes correct and separate accesses

- URI to DB now can be passed as config parameter 'db_uri' or with env variable 'DB_URI',

for example, no need to setup SANIC variables:

```python
create_admin_app(
        host="0.0.0.0", port=os.getenv("PORT", 5000), db=example.models.db, db_models=db_models,
        config={
            "presets_folder": os.path.join(current_path, "csv_to_upload"), "db_uri": "postgresql://local:local@localhost:5432/gino_admin"
        },
    )
```

6. **More fixes**:

- History works again

14.5 Version 0.1.1

1. Fixed annoying UI issues (with icons on buttons & with modal in Init DB page)
2. Fixed some issues with uploading huge Composite CSV

14.6 Version 0.1.0

1. Added REST endpoint to upload data from CSV file to DB.
2. Cleaned up styles in UI.

14.7 Version 0.0.12

1. Now menu in top menu are hidden if you are not authorized
2. Added History logging for actions in Admin panel (edit, delete, add, init_db, load presets and etc) and History page for displaying.
3. Drop DB renamed in Init DB, that better describe feature
4. Fixed deepcopy for models with Integer IDs + other minor issues
5. In UI added normal Display for Bool properties - with check boxes
6. Added Calendar (date & time) pickers in UI for Datetime fields.

14.8 Version 0.0.11:

1. Added possibility to define custom route to Gino Admin Panel. With 'route=' config setting By default, used '/admin' route

2. Added Demo Panel **'Gino-Admin demo'_** - you can log in and play with it. Login & pass - admin / 1234 If you don't see any data in UI maybe somebody before you cleaned it - go to Presets and load one of the data presets.

3. Fixed minors issues: 1)floats now displayed with fixed number of symbols. Parameter can be changed with config param *round_number*=. 2) now file upload fill not raise error if no file was chosen

4. Deepcopy now ask id - you can use auto-generated or define own id to 'deepcopy object'

14.9 Version 0.0.10 Updates:

1. GinoAdmin Config moved to Pydantic. Added possible to send any properties to config with config dict.

2. Added Config param 'name' - this is a name, that will be showed in header near menu. By Default it is display "Sanic-Gino Admin Panel", now you can change it to your header.

3. UI updates: Gino Admin Panel version now showed in UI footer, Login page now more presentable, changed index page of Admin Panel, now it presented main feature.

4. Initialised first project's docs
5. Edit/Delete now take object's unique key as argument and stop fall if in key was '/' symbol

6. Added param 'csv_update_existed' in Config. By default 'csv_update_existed = True'. This mean if you upload CSV with rows with unique keys, that already exist in DB - it will update all fields with values from CSV. You can turn off it with set 'csv_update_existed = False'.

14.10 Version 0.0.9 Updates:

1. Added New feature: REST API to load DB Presets with token auth. Routes: admin/api/auth, admin/api/presets, admin/api/drop_db
2. New feature: Base Cli interface. Was added command *gino_admin run*

14.11 Version 0.0.8 Updates:

1. Added more possibilities to use Gino Admin with applications in different frameworks (Fast API, or aiohttp, or any others)

1.1 Added example how to Gino Admin if main application developed with different Framework (Fast API or smth else). Example in **examples/use_with_any_framework_in_main_app** 1.2 added **create_admin_app** method to full init admin app as separate server 1.3 Old example moved to **base_example/** folder 1.4 in method 'init_admin_app' argument 'gino_models' was renamed to 'db_models'

2. Added support for Unique columns that used in models to identify data row. Previous, your model must have 'id' column for correct work copy/edit/delete methods, but now required ANY unique column in table

Admin Panel checks 'unique' flag in the column. And first unique column will be used to define that row to delete/edit/or copy

If model does not have 'unique' column - it will not showed in admin panel and you will see error message about it in logs as warning.

3. Added display max len of fields in 'Add & Edit' forms

4. New feature "Composite CSV upload" 4.1 Added **Feature "Composite CSV data upload"** - possibility to define one CSV files, that contains several relative tables. Used special to prepare dataset for demo purposes or tests. When it more effective and fast to define relative data in one file. 4.2 Added new config param **composite_csv_settings** that allow to describe some patterns how must be parsed Composite CSV files. Check more information in example and doc's section **Config** 4.3 Example with CSVs samples added to 5. Fixed issue with Logout. 6. Added page 'Settings' to check that Settings are used in admin panel. Display now composite_csv param & presets folder. 7. Added New Feature "Deepcopy" - recursive copy object and all objects, that depend on it.

14.12 Version 0.0.7 Updates:

1. Fixes: datetime issue in 'Copy' action, delete all modal
2. New feature "Presets" (define multiple CSV files with data - upload all with one click).
3. New feature "Drop DB" (full clean up & recreate tables).

New features can be find under menu with 'Cogs' near 'SQL-Runner' button.

14.13 Version 0.0.6 Updates:

1. Clean up template, hide row controls under menu.
2. Added 'Copy' option to DB row.
3. Now errors showed correct in table view pages in process of Delete, Copy, CSV Upload
4. Added possible to work without auth (for Debug purposes). Set env variable 'ADMIN_AUTH_DISABLE=True'
5. Template updated
6. Added export Table's Data to CSV
7. First version of SQL-query execution (run any query and get answer from PostgreSQL)
8. Fixed error display on csv upload

14.14 Version 0.0.5 Updates

1. Upload from CSV: fixed upload from _hash fields - now in step of upload called hash function (same as in edit, or add per item)
2. Fixed errors relative to datetime fields edit, added datetime_str_formats field to Config object, that allows to add custom datetime str formats. They used in step of convert str from DB to datetime object.
3. Now '_hash' fields values in table showed as '*******'
4. Fixed errors relative to int id's. Now they works correct in edit and delete.
5. Update Menu template. Now if there is more when 4 models - they will be available under Dropdown menu.

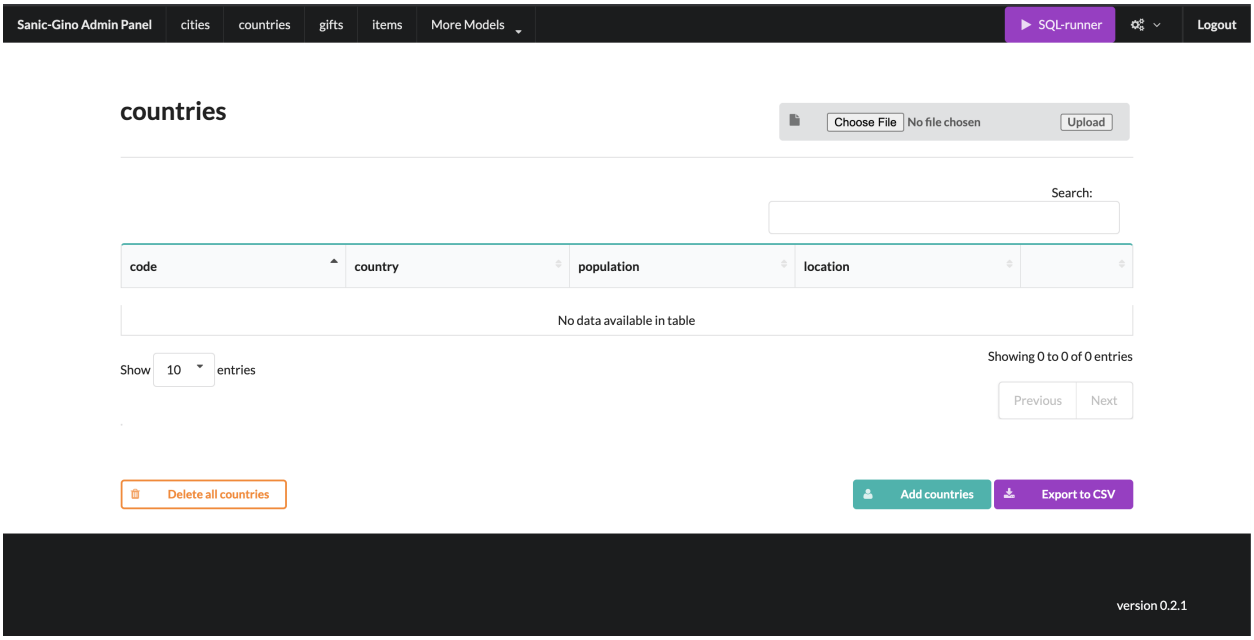
14.15 Version 0.0.4 Updates:

1. Upload from CSV - works, added example to *examples/* files. You can upload data from '.csv' tables.
2. Edit per row - now exist button 'edit'.
3. Fixed delete for ALL rows of the model
4. Fixed delete per element.
5. Now works full 'CRUD'.
6. Fixed auth, now it sets 'cookie' and compare user-agent (for multiple users per login)

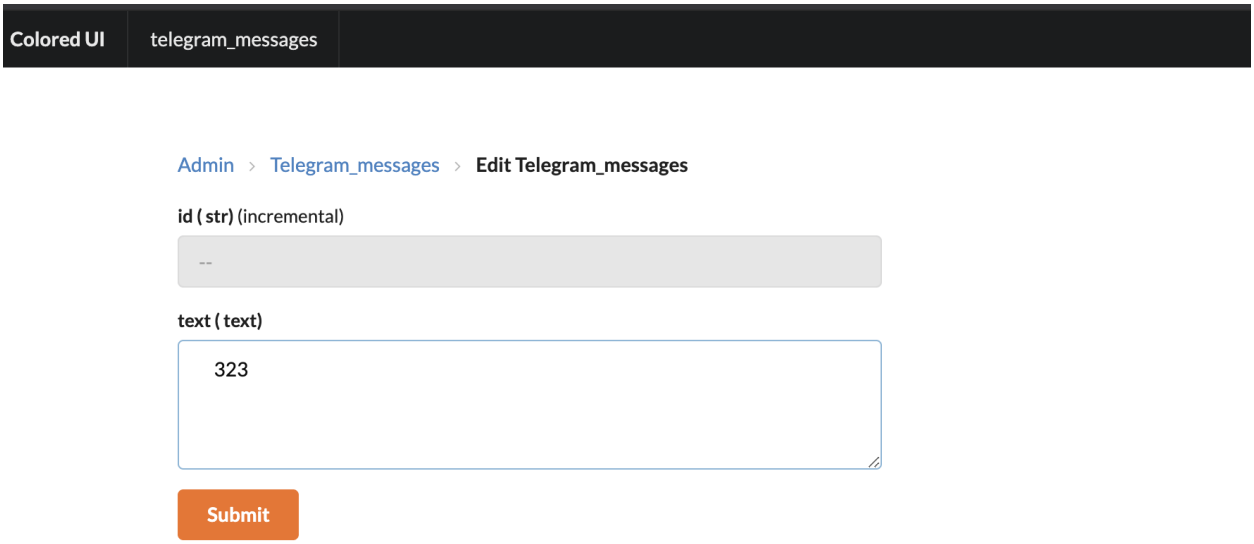
CHAPTER 15

UI Screens

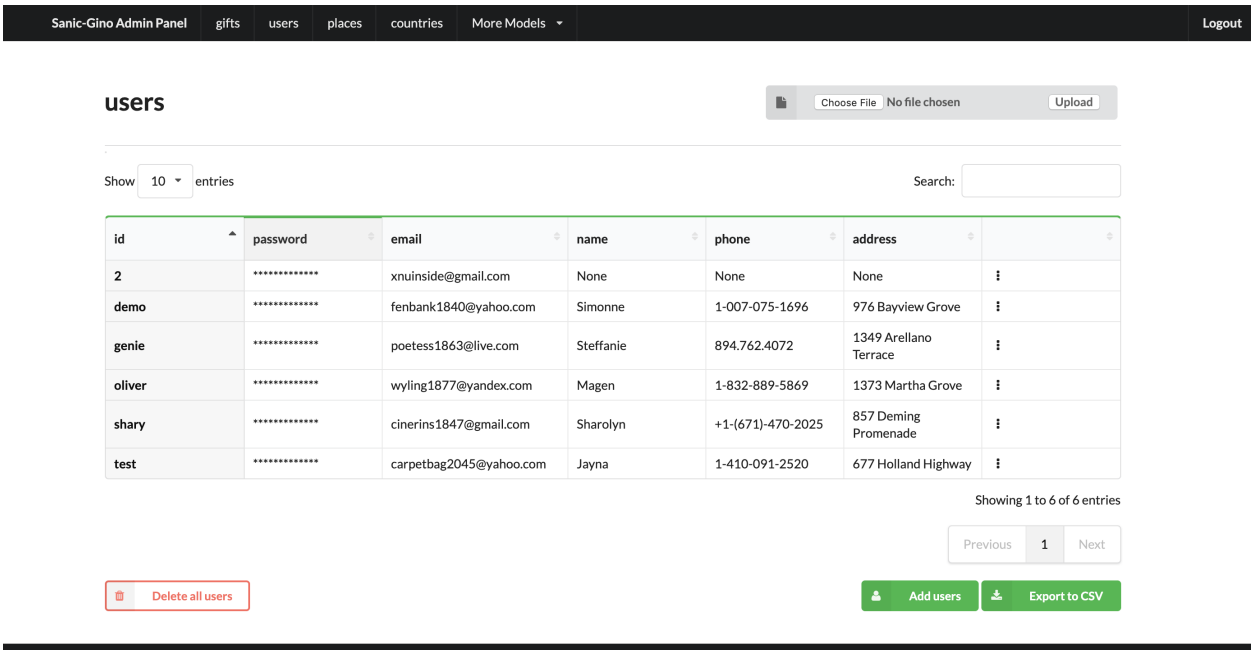
15.1 New colors & possible change UI (since 0.2.1)



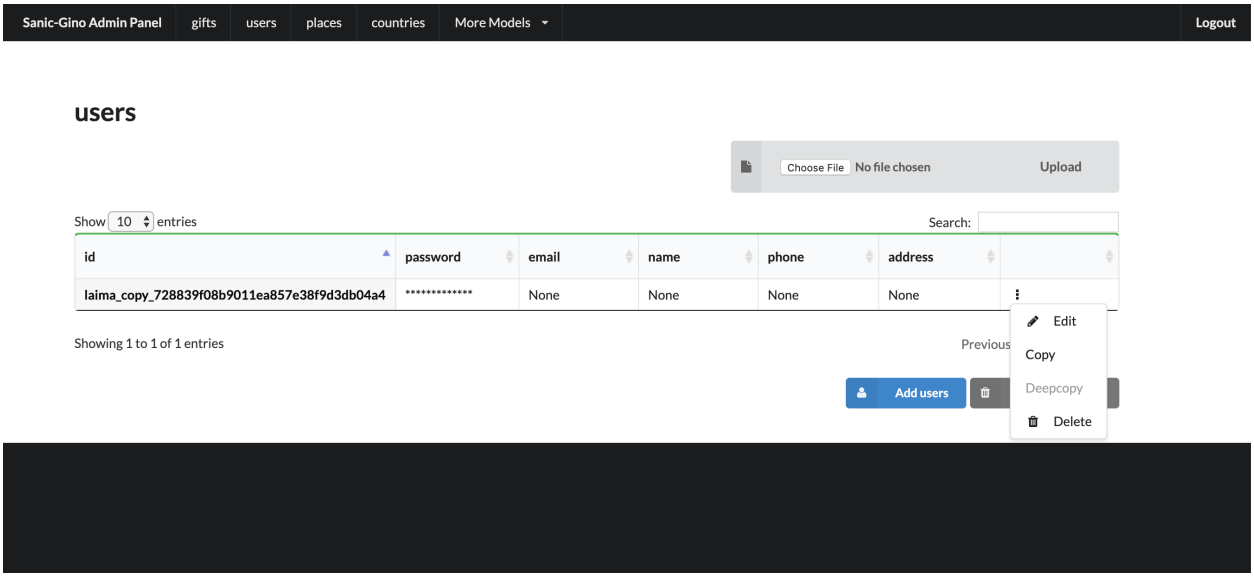
15.2 Incremental Fields



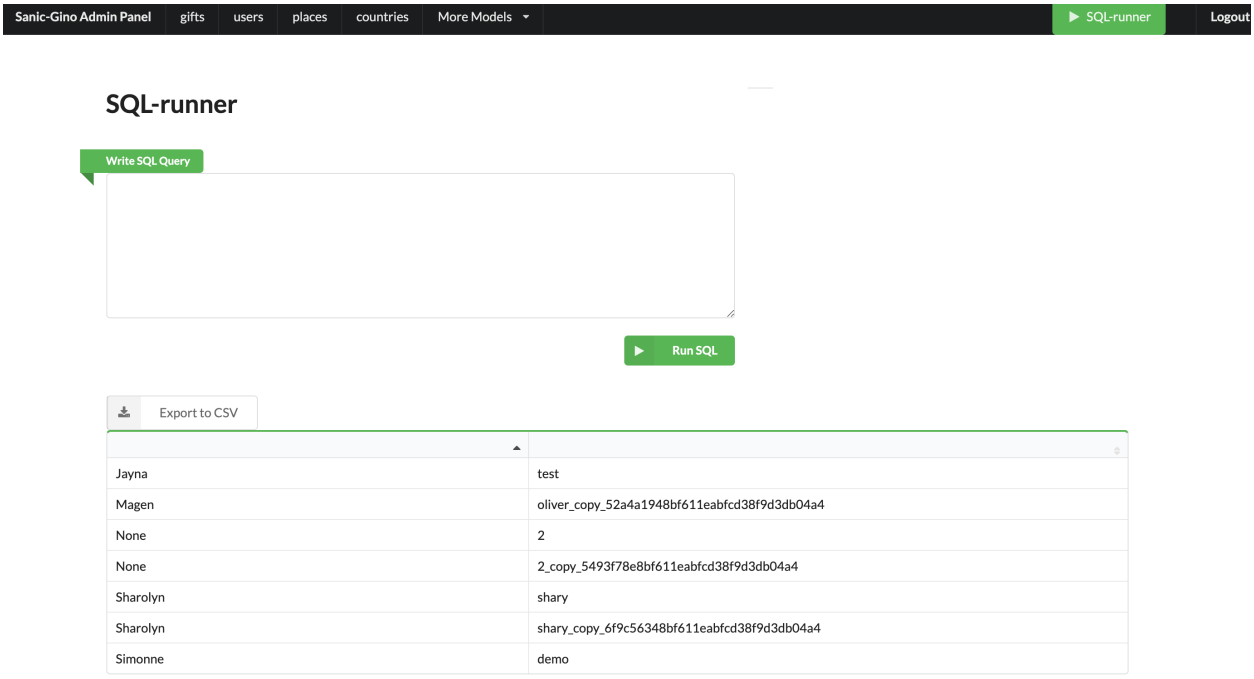
15.3 Table view



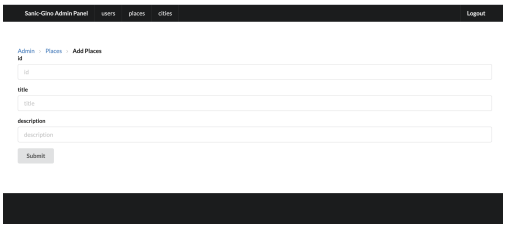
15.4 Edit/delete/copy/deepcopy per item



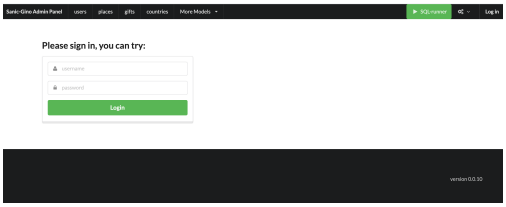
15.5 SQL-runner



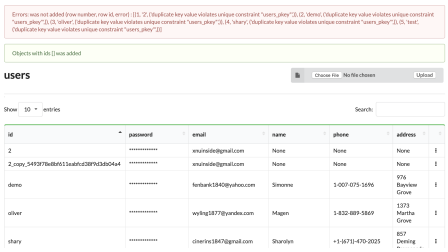
15.6 Add item



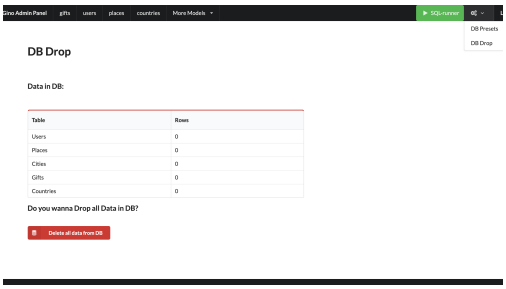
15.7 Authorisation



15.8 Upload data from CSV



15.9 Init DB (drop + recreate tables)



CHAPTER 16

Authorization

1. To disable authorisation:

Set environment variable 'ADMIN_AUTH_DISABLE=1'

```
os.environ['ADMIN_AUTH_DISABLE'] = '1'
```

or from shell:

```
export ADMIN_AUTH_DISABLE=1
```

2. To define admin user & password:

check example/ folder to get code snippets

```
app = Sanic()

app.config["ADMIN_USER"] = "admin"
app.config["ADMIN_PASSWORD"] = "1234"
```