**CHURCH MANAGEMENT API**

# Metadata

Project Name: gnews (good news)

Development started on: 15th/August/2022

Database: MySQL database

Database name: gnews

Programming Language: Python 3.8.5

Backend framework: Django version 4.1

API framework: Django-REST framework

Authentication type: Token authentication

# Developers

## 15TH/AUGUST/2022

1. Francis Murimi Mwangi

DESIGN OF THE PROJECT

The project has been divided into different applications. Each application has been designed to hold different database tables that have been grouped based on how queries will be applied by the API.

The project is defined through the Django Framework.

The virtual environment name is env, and the project was declared using the admin command:

Django-admin startproject gnews

# Applications

This are applications made by running the Django command python manage.py startapp <app-name>

## **2.1 Location**

This application holds the tables that describe geographical locations that will we used to describe where churches and other geographical locations that will be used.

### 2.1.1Model classes (models.py)

1. County

Attributes:

* Name (characters)

1. Sub-county

Attributes:

* county (foreign key)
* name (characters)

1. Town

Attributes:

* county (foreign key)
* sub-county (foreign key)
* name (characters)

Each of the above classes is registered to the admin interface through the admin.py file.

## **2.2 Profile**

This application holds the tables that describe user-accounts that will be used to describe and access different tables within the API queries to the database.

### 2.2.1 Model classes (models.py)

1. User profile

Attributes:

* Profile-owner (one to one)
* Created on (date-time)
* Updated (date-time)
* First-name (characters)
* Last-name (characters)
* Date-of -birth (date-time)
* Gender (character-choice)

1. Pastor Profile

Attributes:

* Profile-owner (one to one)
* Created-on (date-time)
* Updated (date-time)
* First-name (characters)
* Last-name (characters)
* Date-of -birth (date)
* Gender (characters - choice)

1. Leader Profile

Attributes:

* Profile-owner (one to one)
* Created-on (date-time)
* Updated (date-time)
* First-name (characters)
* Last-name (characters)
* Date-of -birth (date)
* Gender (characters-choice)

Each of the above classes is registered to the admin interface through the admin.py file.

### 2.2.2 Serializer classes (serializer.py)

Each model hosted in this application has its serializing classes in this file.

1. User-profile-serializer

Class model: user-profile

Fields:

* id
* first-name
* last-name
* gender
* date-of-birth

1. Pastor-profile-serializer

Class model: user-profile

Fields:

* id
* first-name
* last-name
* gender'
* date-of-birth

1. Leader-profile-serializer

Class model: user-profile

Fields:

* id
* first-name
* last-name
* gender
* date-of-birth

## **2.3 church**

This application holds the classes that are used to describe and abstract tables that will be used to hold information about church types (Protestants, etc.), denominations, individual churches, church pastors, denomination leaders and church memberships.

### Model classes (models.py)

1. Church Type

Attributes:

* Title (character field)
* Description (text field)

1. Church Denomination

Attributes:

* Full title (character field)
* Title (character field)
* Description (text field)
* Church type (foreign key)
* Head-quarters (foreign key- town)
* Leader (many to many)

1. Denomination Leadership

Attributes:

* denomination (foreign key)
* leader profile (foreign key)
* created on (date-time)
* updated (date)

1. Church item

Attributes:

* Title (character)
* Location (foreign key)
* Church Denomination (foreign key)
* Pastors (many to many)
* Members (many to many)

1. Church membership

Attributes:

* Church (foreign key)
* Use profile (foreign key)
* Created on (date)

1. Church pastorship

Attributes:

* Church (foreign key)
* Use profile (foreign key)
* Created on (date)

Each of the above classes is registered to the admin interface through the admin.py file.

## 2.3.2 **Serializers (serializer.py)**

1. **Church-Item-Serializer (Model-Serializer)**

Model: Church-Item

Fields: all

1. **Church-Serializer (ModelSerializer)**

This class is used to serialize church item in denomination list

Model: Church-Item

Fields:

* Id
* Title
* Church-denomination
* Pastor

1. **Church-Denomination-Serializer (Model-Serializer)**

Churches related to a specific denomination are pre-fetched

Model: Church-Denomination

Fields:

* Id
* Title
* Full title
* Leader
* Headquarters
* Church type
* churches

1. **Church-Denomination-Leadership-Serializer (Model-Serializer)**

Model: Denomination-Leadership

Fields: all

1. **Church-Membership-Serializer (Model-Serializer)**

Model: Church-Membership

Fields:

* Id
* Church
* Created-on
* Comment

1. **Add-Membership-Serializer (Model-Serializer)**

Model: Church-Membership

Fields:

* comment

1. **Church-Pastorship-Serializer (Model-Serializer)**

Model: Church-Pastorship

Fields: all

## **2.4 Product**

This application holds the models for products that each church offers and associated serializer classes for the models.

### Models (models.py)

1. Topic

Attributes:

* title (characters)
* slug (slug field)
* Created-by (foreign key)
* Church (foreign key)
* description (text field)
* created (date-time)
* updated (date-time)

1. Service

Attributes:

* Church (foreign key)
* Title (characters)
* Service day (characters choices)
* Start time (time field)
* End time (time field)
* Description (text field)
* Created (date time)
* Updated (date time)

1. Blog

Attributes:

* Category (many to many field for topics)
* Created by (foreign key user-profile)
* Service (foreign key)
* Church (foreign key)
* Pastor (foreign key pastor-profile)
* Title (character field)
* Slug (slug field)
* Status ( integer field, choices)
* Created (date time field)
* Updated (date time field)

1. Comment

Attributes:

* Blog (foreign key)
* Created by (foreign key, user profile)
* Content (text field)
* Created on (date time)
* Active (Boolean field)

### Serializers (serializers.py)

It contains the serializer classes that will be used to process the product models data in the API.

1. Comment-Serializer (serializers.ModelSerializer):

Model: Comment

Fields:

* Id
* Created-on
* Content

1. Blog-Serializer (serializers.ModelSerializer):

Comments: Comment-Serializer

Model: Blog

Fields:

* Id
* Category
* Pastor
* Church
* Service
* Title
* Content
* Created
* Comments

1. Blog-List-Serializer (serializers.ModelSerializer):

To use with service serializer, topic serializer, pastor list

Model: Blog

Fields:

* Id
* Title
* Created

Service-Serializer (serializers.ModelSerializer):

Model: Service

Fields:

* Id
* Title
* Service-day
* Start-time
* End-time
* Description

1. Service-Blog-Serializer (serializers.ModelSerializer):

Blogs: BlogListSerializer

Model: Service

Fields:

* Id
* Title
* Service-day
* Start-time
* End-time
* Description
* Blogs

Topic-Serializer (serializers.ModelSerializer):

Model: Topic

Fields:

* Id
* Title
* Description

Topic-Blog-Serializer (serializers.ModelSerializer):

Blogs: Blog-List-Serializer

Model: Topic

Fields:

* Id
* Title
* Description
* Blogs

API DESIGN

# Introduction

The API has been hosted in an application called apiv1. This is meant to make it easy to choose the version of API being used and is available for consumption.

The files in is application are ordered the same way other files in other applications are ordered.

# API files

## Serializers (serializers.py)

This holds serialize classes for models that come already set within the Django framework.

### Userserializer class

Model: User

Fields:

* username
* email
* Password

Class methods:

* Create (). This methods creates users and saves them giving each created user a token that is used for authentication.

## **Views (views.py)**

This file holds the views that are used to process data within the API. Each view is a class.

### User-Create (Create-APIView)

This class allows users to be created. It is exempted from authentication and no permission is required to access this class.

Serializer: User-serialize

### Login-View (APIView)

This class allows created users to log in. It provides the authentication token unique for each user. No permission is required to access this class.

Serializer: User-serialize

### User-Profile-List (ListCreateAPIView)

This class allows a user to add and view profiles associated with his user account.

Serializer: user-profile-serialize

Class methods:

* Get-queryset
* Perform-create

### User-Profile-Detail (RetrieveUpdateDestroyAPIView)

This class allows a user to view, update and delete one user profile.

Serializer: user-profile-serialize

Class methods:

* Get-queryset
* Update
* Destroy

### Pastor-Profile-List (ListCreateAPIView)

This class allows a user to add and view pastor profiles associated with his user account.

Serializer: pastor-profile-serialize

Class methods:

* Get-queryset
* Perform-create

### Pastor-Profile-Detail (RetrieveUpdateDestroyAPIView)

This class allows a user to view, update and delete one pastor profile.

Serializer: pastor-profile-serialize

Class methods:

* Get-queryset
* Update
* Destroy

### Leader-Profile-List (ListCreateAPIView)

This class allows a user to add and view leader profiles associated with his user account.

Serializer: leader-profile-serialize

Class methods:

* Get-queryset
* Perform-create

### Leader-Profile-Detail (RetrieveUpdateDestroyAPIView)

This class allows a user to view, update and delete one leader profile.

Serializer: leader-profile-serialize

Class methods:

* Get-queryset
* Update
* Destroy

## **Church views (church\_views.py)**

This file holds the views that process information and data about church models.

### Denomination-List (ListAPIView)

This view lists all the available denominations and gives a list of churches associated with that denomination.

Serializer: Church-Denomination-Serializer

Class methods:

* Get queryset

### Church-List (ListAPIView)

This view lists all the churches.

Serializer: Church-Item-Serializer

Class methods:

* Get queryset

### Church-Detail (RetrieveAPIView)

This view shows details of a single church item.

Serializer: Church-Item-Serializer

Class methods:

* Get-queryset

### Church-Membership-List (ListAPIView)

This view shows a list of all the churches that the user is a member.

Serializer: Church-Membership-Serializer

Class methods:

* Get-queryset

### Church-Membership-Detail (RetrieveUpdateDestroyAPIView)

This view shows the details of the membership of the user to a given church.

Serializer: Church-Membership-Serializer

Class methods:

* Get queryset
* Update
* Destroy

### Add-Church-Membership (CreateAPIView)

This view allows the user to add a church membership to a church.

Serializer: Add-Membership-Serializer

Class methods:

* Get queryset
* Perform create

## **Product views (product\_views.py)**

This are the view that process data, add, delete, update and create within product models.

### TopicList (generics.ListCreateAPIView):

This view lists topics of a given church.

Serializer class: TopicSerializer.

Class methods:

* Get-query set
* Perform-create

### TopicDetail (generics.RetrieveUpdateAPIView):

This view shows details of a single topic. It also lists blogs associated with the selected topic. It also allows for the topic to be updated.

Serializer class: TopicSerializer

Class methods:

* Get query set
* Update

### ServiceList (generics.ListCreateAPIView):

This view shows the list of all services associated with a single church. It also allows the creation of a new service.

Class serializer: ServiceSerializer

Class methods:

* Get query set
* Perform create

### ServiceDetail (generics.RetrieveUpdateAPIView):

This view shows details of a selected service, allows for the update of the selected service and also shows blogs associated with the service.

Class serializer: ServiceSerializer

Class methods:

* Get query set
* Update

### ChurchPastorList (generics.ListAPIView):

This view shows a list of all pastors associated with a given church.

Serializer class = PastorProfileSerializer

Class methods:

* Get-query set

### ChurchPastorBlogList (generics.ListAPIView):

This view shows a list of all blogs associated with a given pastor for a given church.

Serializer class: BlogListSerializer

Class methods:

* Get query set

### ChurchBlogList (generics.ListCreateAPIView):

This view shows a list of all blogs associated with a specific view. It also allows for creation of a new blog that belongs to that church.

Serializer class: BlogSerializer

Class methods:

* Get query set
* Perform create

### BlogDetail (generics.RetrieveUpdateAPIView):

This view shows details of a given blog. It also allows for the update of the selected blog and shows comments associated with that blog.

Serializer class: BlogSerializer

Class methods:

* Get query set
* Update

### CommentList (generics.ListCreateAPIView):

This view shows a list of all comments associated with a given blog and also allows creation of a new comment.

Serializer class: CommentSerializer

Class methods:

* Get query set
* Perform create

### CommentDetail (generics.RetrieveUpdateDestroyAPIView):

This view shows details of a given comment. It also allows the update and deletion of a selected comment.

Serializer class: CommentSerializer

Class methods:

* Get query set
* Update
* Destroy

## Urls (urls.py)

This is the file that holds the API endpoints.

They are:

1. path("user/", UserCreate.as\_view()) user creation
2. path('login/',LoginView.as\_view()), login
3. path('profile/',UserProfileList.as\_view()), user profile list
4. path('profile/<int:pk>/',UserProfileDetail.as\_view()), user profile detail
5. path('pastor/',PastorProfileList.as\_view()), pastor profile list
6. path('pastor/<int:pk>/',PastorProfileDetail.as\_view()), pastor profile detail
7. path('leader/',LeaderProfileList.as\_view()), leader profile list
8. path('leader/<int:pk>/',LeaderProfileDetail.as\_view()), leader profile detail
9. path('denomination/',DenominationList.as\_view()), denomination list
10. path('church/',ChurchList.as\_view()), church list
11. path('church/<int:pk>/',ChurchDetail.as\_view()), church details
12. path('member/',ChurchMembershipList.as\_view()), church membership list
13. path('member/<int:pk>/',ChurchMembershipDetail.as\_view()), church membership detail
14. path('church/<int:pk>/member/',AddChurchMembership.as\_view()), add a church membership
15. path('church/<int:pk>/topic/',TopicList.as\_view()), topic list
16. path('topic/<int:pk>/',TopicDetail.as\_view()), topic detail
17. path('church/<int:pk>/service/',ServiceList.as\_view()), services list of a given church
18. path('service/<int:pk>/',ServiceDetail.as\_view()), service detail
19. path('church/<int:pk>/pastor/',ChurchPastorList.as\_view()) List of pastors in a given church
20. path('church/<int:pk>/pastor/<int:id>/',ChurchPastorBlogList.as\_view()), List of blogs by a pastor in a select church
21. path('church/<int:pk>/blog/',ChurchBlogList.as\_view()), list of blogs for a given church
22. path('blog/<int:pk>/',BlogDetail.as\_view()),blog detail
23. path('blog/<int:pk>/comment/',CommentList.as\_view()), comments for a specific blog
24. path('comment/<int:pk>/',CommentDetail.as\_view()),comment detail