Guided Exploration 03

Bits-and-Bytes: Emiliano, Darion, Matt, Chris, and Ken

## Summary of this Document:

This document holds technical documentation and resources that help with understanding and implementing the areas of focus.

# GE-03 Key areas of focus:

1. [Setting up Database and Queries](#_mvferxyquhj3) - Ken
2. [SuperUser and Admin Panel](#_4e1cqwu8j26a) - Darion
3. [BDD](#_q7qo4kl1hmi9) - Chris
4. [Models and Relationships](#_olwz3hsy0lj9) - Emiliano
5. [REST & CRUD](#_41nfv4vu3a73) - Matt

# Setting Up Database and Queries

python manage.py makemigrations

#Apply the migrations

python manage.py migrate

#Verify the migrations

python manage.py dbshell

#List the migrations

./manage.py showmigrations

<https://docs.djangoproject.com/en/4.2/intro/tutorial02/>

This is already done at last GE, just in case. I wrote some migrations command and a link to how to setup the database if they has no done it before.

**Source:** <https://docs.djangoproject.com/en/4.2/topics/migrations/>

* This is a source that talks about migrations specifically. There are specific commands that let the user make a migration and actually migrate the database. This source was used to find out what generated the files once we made the migration.

**Source:** [**https://docs.djangoproject.com/en/4.2/ref/models/querysets/**](https://docs.djangoproject.com/en/4.2/ref/models/querysets/)

# SuperUser and Admin Panel

- python manage.py createsuperuser

- Follow prompts setting up user access information (username and password)

- visit http://127.0.0.1:8000/admin

- login with superuser credentials

# 

# BDD *(Behavior Driven Development)*

**Sources:**

1. <https://cucumber.io/docs/bdd/>
2. <https://www.geeksforgeeks.org/behavioral-driven-development-bdd-in-software-engineering/>

Behavior Driven Design is a process of creating software through various steps. These include working with the users and creating user stories/scenarios, documenting these in preparation for automation, and then finally implementing them into an iteration, all while remaining in contact with the users and running tests. The purpose is to efficiently work through projects with the users to reduce the amount of time backtracking.

# 

# 

# 

# Models and Relationships

**Source:** <https://docs.djangoproject.com/en/4.2/topics/db/examples/>

⁃ This is directly from the Django website, which outlines three examples of the three different relationships. This technical documentation shows the many-to-many, one-to-many, and one-to-one. Each example has its own scenario and outlines the relationship in code snippets with explanations.

**Source:** <https://www.geeksforgeeks.org/django-models/>

⁃ The Geeks for Geeks website explains more about the concepts of models and relationships. There is a lot of overlapping information with the focus areas, which makes this document valuable to use. Some important things to note are the data types and fields that are used in Django. These types and fields are located in the table with direct links to what each data type and field accomplishes.

**Source:** <https://www.w3schools.com/django/django_models.php>

⁃ W3 schools have a rudimentary approach to models in Django. This is a website with simplified explanations and step-by-step guides to creating a model.

## Models and Active Record Pattern

**Source:** <https://docs.djangoproject.com/en/4.2/topics/db/models/>

* This documentation was used to learn more about models and how they relate to the active record pattern.

**Source:**<https://dev.to/andrewsavetchuk/overview-of-the-active-record-pattern-2jdo>

* This was to learn about the general design of the active record pattern. The active record pattern is not a Django-only concept but it is used outside in other frameworks. This resource was mainly used in the GE03 learning where it complimented the formal Django documentation.

# REST and CRUD

**CRUD Paradigm**

- Create, Read, Update, Delete

- Abstract Concept for creating APIs

- For building APIs, models need to provide four basic types of functionality. The model

must be able to Create, Read, Update, and Delete resources.

- Create > POST

- Read > GET

- Update > PUT

- Delete > DELETE

**REST**

- REpresentational State Transfer

- Architectural style for providing standards between computer systems on the web

- Separation of Client and Server

o clients send requests to retrieve or modify resources, and servers send responses to these requests.

- REST requires that a client make a request to the server to retrieve or modify data on the server. A request consists of:

o an HTTP verb, which defines what kind of operation to perform

o a header, which allows the client to pass along information about the request

o a path to a resource

o an optional message body containing data

- HTTP Verbs

- 4 basic HTTP verbs used in requests to interact with resources in a REST system:

o GET — retrieve a specific resource (by id) or a collection of resources

o POST — create a new resource

o PUT — update a specific resource (by id)

o DELETE — remove a specific resource by id

# 