Database setup

Now, open up `mysite/settings.py`. It's a normal Python module with module-level variables representing Django settings.

By default, the configuration uses SQLite. If you're new to databases, or you're just interested in trying Django, this is the easiest choice. SQLite is included in Python, so you won't need to install anything else to support your database. When starting your first real project, however, you may want to use a more scalable database like PostgreSQL, to avoid database-switching headaches down the road.

If you wish to use another database, install the appropriate database bindings and change the following keys in the `DATABASES` `'default'` item to match your database connection settings:

- `ENGINE` – Either `'django.db.backends.sqlite3'`, `'django.db.backends.postgresql'`, `'django.db.backends.mysql'`, or `'django.db.backends.oracle'`. Other backends are also available.
- `NAME` – The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, `NAME` should be the full absolute path, including filename, of that file. The default value, `BASE_DIR / 'db.sqlite3'`, will store the file in your project directory.

If you are not using SQLite as your database, additional settings such as `USER`, `PASSWORD`, and `HOST` must be added. For more details, see the reference documentation for `DATABASES`.

> For databases other than SQLite
>
> If you're using a database besides SQLite, make sure you've created a database by this point. Do that with "`CREATE DATABASE database_name;`" within your database's interactive prompt.
>
> Also make sure that the database user provided in `mysite/settings.py` has "create database" privileges. This allows automatic creation of a test database which will be needed in a later tutorial.
>
> If you're using SQLite, you don't need to create anything beforehand - the database file will be created automatically when it is needed.

While you're editing mysite/settings.py, set TIME_ZONE to your time zone.

Also, note the INSTALLED_APPS setting at the top of the file. That holds the names of all Django applications that are activated in this Django instance. Apps can be used in multiple projects, and you can package and distribute them for use by others in their projects.

By default, INSTALLED_APPS contains the following apps, all of which come with Django:

django.contrib.admin – The admin site. You'll use it shortly.
django.contrib.auth – An authentication system.
django.contrib.contenttypes – A framework for content types.
django.contrib.sessions – A session framework.
django.contrib.messages – A messaging framework.
django.contrib.staticfiles – A framework for managing static files.
These applications are included by default as a convenience for the common case.

Some of these applications make use of at least one database table, though, so we need to create the tables in the database before we can use them. To do that, run the following command:

```
$ python manage.py migrate
```

The `migrate` command looks at the `INSTALLED_APPS` setting and creates any necessary database tables according to the database settings in your `mysite/settings.py` file and the database migrations shipped with the app (we'll cover those later). You'll see a message for each migration it applies. If you're interested, run the command-line client for your database and type `\dt` (PostgreSQL), `SHOW TABLES;` (MariaDB, MySQL), `.tables` (SQLite), or `SELECT TABLE_NAME FROM USER_TABLES;` (Oracle) to display the tables Django created.

For the minimalists

Like we said above, the default applications are included for the common case, but not everybody needs them. If you don't need any or all of them, feel free to comment-out or delete the appropriate line(s) from `INSTALLED_APPS` before running `migrate`. The `migrate` command will only run migrations for apps in `INSTALLED_APPS`.

Revision #1
Created 22 February 2023 17:40:49 by Admin
Updated 22 February 2023 18:01:43 by Admin