Getting Started with Models

In this lesson, we will find out how to create models in our Flask application.

WE'LL COVER THE FOLLOWING

Introduction

You might recall that at the beginning of this course, we learned about three **components** of a web application:

1. **Views**
2. **Templates**
3. **Models**

In previous lessons, we have covered two of these components (**views** and **templates**). In this lesson, we will figure out how to create **models**. Later in the course, we will also get to know how to manipulate these models.

Steps to create a `User` model in a database

Again, let's get help from the example we have been using. We used a dictionary to store the user's data.

```
users = {
    "archie.andrews@email.com": "football4life",
    "veronica.lodge@email.com": "fashiondiva"
}
```

But this is poor application logic. Therefore, to store the user data inside the database, we first need to create models. The models will define the schema for the database table, its properties, and its functionalities.

Create a model class for `User`

First, we will create a class for the `User` table. This class will inherit from the model class from SQLAlchemy.

```
class User(db.Model):
```

Add columns/attributes

Next, we will add the columns for the table. In the example, the users only had two fields: email and password. So we will add them. Each column will be an object of the `Column` subclass of `SQLAlchemy`.

```
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)
```

In the snippet given above you can observe that we have given some additional parameters to the columns. The **first parameter** of the column defines the data type for the column. Following are the allowed class names:

1. **`Integer`**
2. **`String(size)`**
3. **`Text`**
4. **`DateTime`**
5. **`Float`**
6. **`Boolean`**
7. **`PickleType`**
8. **`LargeBinary`**

There are many optional parameters as well; some of the most commonly used ones are given below:

- **`primary_key`**: If set `True`, it will indicate the primary key of the table.

- `primary_key`: If set `True`, it will indicate the primary key of the table.

- `nullable`: If set `False`, it will be compulsory to set the value for that column.
- `unique`: If this parameter is set to `True` then all values for this column are checked for uniqueness.
- `index`: Setting it to `True` indicates that this column will be indexed.

Creating a table in a database

Now, with a simple command, the model that we just made will be created in the database.

```
db.create_all()
```

> **Important Note:** The `create_all()` command should be called **after** the models are declared. If the models are defined in a separate module, then they should be imported first. Furthermore, during development, if new models are created, then this command **only creates the newly defined models**.

Complete implementation

Let’s add this implementation to the previously discussed example.

> **Note:** You will notice that we have not yet removed the dictionary from this example. This is because we have yet to learn how to retrieve and manipulate the data from the **models**.

```
#header {
  padding: 30px;
  text-align: center;
  background: red;
  color: white;
  font-size: 40px;
}
#footer {
   position: fixed;
   width: 100%;
   background-color: #BBC4C2;
   color: white;
```

```
    text-align: center;
    left: 0;
    bottom:0;
}
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

li {
  display: inline;
}
```

In the next lesson, we will learn about creating relationships among models in our application. We will start with the *“one-to-many”* relationship first.