New chat

Help

Activity

Settings

Egypt   
From your IP address • Update location

Gemini Advanced

**Conversation with Gemini**



hi gemeni, i have a microsoft sql database with 9 tables and i want to make a web application with a list of users for them to search and get filtered results from those tables, the site can run locally (they can access the sql database through network already) or we can host it online doesnt matter, so i guess we will go with django right? i only heard bout it so you will help me out if we go with django

Show drafts

Hi there! Yes, Django is an excellent choice for your web application. It's a powerful and versatile framework that makes building complex web applications like yours a lot easier. Here's a breakdown of why it's a good fit and how it can help you:

**Why Django is a Good Choice:**

* **Built for Data-Driven Applications:** Django's ORM (Object-Relational Mapper) makes it seamless to interact with your SQL database. You can define your models (representing your 9 tables) in Python, and Django handles all the SQL queries behind the scenes.
* **Robust and Secure:** Django comes with built-in features for security, user authentication, and administration, which are crucial for any web application.
* **Scalable:** Django's architecture allows you to scale your application as it grows and handles more users and data.
* **Large Community and Resources:** Django has a massive community and tons of documentation, tutorials, and third-party libraries to help you at every step.

**How Django Can Help:**

1. **Models:** You'll define models for each of your 9 tables. This will create a clear structure for your data and allow you to interact with it easily.
2. **Views:** You'll create views to handle user requests (searching and filtering data) and render the results into HTML templates.
3. **Templates:** You'll use templates to create the look and feel of your web application.
4. **Forms:** You'll likely use forms to collect search criteria from your users.
5. **Authentication:** You'll implement user authentication and authorization to control access to your data.

**Next Steps:**

1. **Install Django:** If you don't have it already, you'll need to install Django and set up a new project.
2. **Define Models:** Create models representing your 9 tables.
3. **Create Views:** Build views to handle user requests (searching and filtering) and render the results.
4. **Design Templates:** Create HTML templates to display the data to your users.

**Note:** While you can run Django locally, hosting it online might offer advantages like easier access for your users and potential scalability.

I'm here to help you throughout the process! Feel free to ask if you have any more questions or need guidance on specific steps.



ok lets start, by install djago you mean pip install or go to thier site and download something?

Show drafts

You're absolutely right, let's get started! To install Django, we'll use pip install. This is the recommended and most common way to install Django.

Here's what you need to do:

1. **Make sure you have Python installed:** Django is a Python framework, so you'll need Python installed on your system. You can check if you have it by opening your terminal or command prompt and typing python --version. If you don't have it, you can download it from the official Python website: <https://www.python.org/>
2. **(Recommended) Create a virtual environment:** It's a good practice to create a virtual environment for your Django project. This keeps your project's dependencies isolated from other Python projects you might have.
   * To create a virtual environment, you can use the following command in your terminal:

Bash

python -m venv myprojectenv

Use code [with caution.](/faq#coding)

* + Replace myprojectenv with the name you want to give your virtual environment.
  + Activate the virtual environment:
    - On Windows:

Bash

myprojectenv\Scripts\activate

Use code [with caution.](/faq#coding)

* + - On macOS/Linux:

Bash

source myprojectenv/bin/activate

Use code [with caution.](/faq#coding)

1. **Install Django:** Once your virtual environment is activated, you can install Django using pip:

Bash

pip install django

Use code [with caution.](/faq#coding)

This will download and install the latest stable version of Django.

That's it! You've successfully installed Django and are ready to start building your web application.

Feel free to ask if you have any questions or encounter any issues during the installation process. I'm here to help!



i want my django project files to be backed up on my onedrive and i will commit them to my repo, i dont want all the venv files to be backed up, can i set up the djanjo project in one place inside my onedrive and repo but set up the virtual enviroment in another outside onedrive? i think using vscode i can choose the interpeter but it doesnt let me browse for it

Show drafts

Yes, you can absolutely set up your Django project in one place within your OneDrive and repository while keeping the virtual environment separate outside of OneDrive. This is a good practice as it keeps your project files organized and prevents unnecessary backup of the virtual environment files.

Here's how you can achieve this:

1. **Create Django project within OneDrive:**
   * Choose a location within your OneDrive where you want to store your project files.
   * Open your terminal or command prompt and navigate to that location.
   * Create your Django project using the following command:

Bash

django-admin startproject myproject

Use code [with caution.](/faq#coding)

* + Replace myproject with the desired name for your project.

1. **Create virtual environment outside OneDrive:**
   * Choose a location outside of OneDrive where you want to store your virtual environments.
   * Navigate to that location in your terminal.
   * Create the virtual environment using the command:

Bash

python -m venv myprojectenv

Use code [with caution.](/faq#coding)

* + Replace myprojectenv with the name you want to give your virtual environment.

1. **Activate virtual environment and install Django:**
   * Activate the virtual environment you just created.
     + On Windows:

Bash

myprojectenv\Scripts\activate

Use code [with caution.](/faq#coding)

* + - On macOS/Linux:

Bash

source myprojectenv/bin/activate

Use code [with caution.](/faq#coding)

* + Install Django within the activated virtual environment:

Bash

pip install django

Use code [with caution.](/faq#coding)

1. **Configure VS Code to use the external virtual environment:**
   * Open your Django project in VS Code.
   * Open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P).
   * Type "Python: Select Interpreter" and select it.
   * In the list of available interpreters, you should see an option to "Enter interpreter path...". Select this.
   * Manually browse to the location of your virtual environment's Python executable. This will usually be within the Scripts folder (on Windows) or the bin folder (on macOS/Linux) of your virtual environment.
   * Select the Python executable.

Now, VS Code will use the Python interpreter from your external virtual environment, ensuring that your project runs within that environment even though the project files are located within OneDrive.

Remember to commit your project files to your repository, but exclude the virtual environment folder from your commits (usually by adding it to your .gitignore file).

This setup allows you to keep your Django project backed up on OneDrive while maintaining a clean separation between your project files and the virtual environment, which is generally good practice.

If you encounter any difficulties with setting up the interpreter in VS Code, feel free to ask for further assistance.



CommandError: 'omar.warehouse' is not a valid project name. Please make sure the name is a valid identifier. , should i put them next to each other or use an underscore?

Show drafts

You should use an underscore to separate the words in your project name, like this: omar\_warehouse.

Django project names need to follow specific rules to be valid identifiers. Here are the key points:

* **Start with a letter or underscore:** The first character must be a letter or an underscore.
* **Contain only letters, numbers, and underscores:** No other special characters are allowed.
* **Cannot be a reserved Python keyword:** Avoid using words like if, else, for, etc., as project names.

Using an underscore is the most common and recommended way to separate words in Django project names. It keeps the name readable and avoids any potential conflicts with reserved keywords or other naming conventions.

So, go ahead and try creating your project again using the name omar\_warehouse:

Bash

django-admin startproject omar\_warehouse

Use code [with caution.](/faq#coding)

This should work without any issues. If you encounter any further problems, don't hesitate to ask!



ok done, now i have a bunch of files, where do i start? before we talk about acccessing the database, should i create 3 users for me and for "nada" and "yasser"?

Show drafts

Excellent! You're making progress. Now that you have your Django project set up, let's get started.

**Where to start:**

The main file you'll interact with initially is manage.py. This file is located in the root directory of your project (omar\_warehouse). It's used to execute various Django commands.

**Creating users:**

Yes, it's a good idea to create users for yourself, Nada, and Yasser before we dive into database access. Django has a built-in user authentication system that makes this process straightforward.

Here's how you can create the users:

1. **Activate your virtual environment:** Make sure you're in the activated virtual environment where you installed Django.
2. **Start the Django shell:**

Bash

python manage.py shell

Use code [with caution.](/faq#coding)

This will open an interactive Python shell where you can execute Django commands.

1. **Create the users:**

Python

from django.contrib.auth.models import User

# Create your user

you = User.objects.create\_user('your\_username', 'your\_email@example.com', 'your\_password')

# Create Nada's user

nada = User.objects.create\_user('nada', 'nada@example.com', 'nada\_password')

# Create Yasser's user

yasser = User.objects.create\_user('yasser', 'yasser@example.com', 'yasser\_password')

Use code [with caution.](/faq#coding)

* + Replace your\_username, your\_email@example.com, and your\_password with your actual username, email, and password.
  + Do the same for Nada and Yasser's information.

1. **Save the changes:**

Python

you.save()

nada.save()

yasser.save()

Use code [with caution.](/faq#coding)

1. **Exit the shell:**

Python

exit()

Use code [with caution.](/faq#coding)

Now you have three users created in your Django project. You can use these users later for authentication and authorization purposes when you build the web application.

**Next steps:**

* **Set up database access:** We'll configure Django to connect to your Microsoft SQL database.
* **Define models:** We'll create Django models to represent your 9 tables in the database.
* **Build views and templates:** We'll create views to handle user requests (search and filter) and templates to display the data.

Feel free to ask if you have any questions along the way. We'll take it step by step!



D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py shell

Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 8.20.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:  , question, why am i in the terminal, shoudnt i make a py file in my project and save my work or adding and creating users is something temporary and i should stay in the terminal? also , can i even paste multiple lines of python code inside the terminal?

Show drafts

You are absolutely correct to wonder about these aspects! Let's clarify a few things:

**Why the terminal?**

* **Temporary tasks:** The Django shell (which you accessed by running python manage.py shell) is ideal for quick, one-off tasks like creating users, testing database queries, or interacting directly with your Django models. It's a convenient way to experiment and get immediate feedback.
* **Not for permanent code:** The shell isn't meant for writing and saving your main application logic. That's where .py files come in.

**Should you create a .py file?**

* **Yes, for application logic:** You will definitely need to create .py files within your Django project to define your models, views, and other components of your web application.
* **Organization:** .py files provide structure and allow you to organize your code into reusable modules.

**Pasting multiple lines in the terminal:**

* **Yes, you can!** Most terminals allow you to paste multiple lines of code.
* **IPython (the enhanced shell you're using) is even better:** IPython has some handy features for multi-line input.
  + You can paste multiple lines directly, and it will usually handle indentation correctly.
  + If you want more control, you can use the %paste magic command to paste code from your clipboard, preserving formatting.
  + You can also use triple quotes (''') or triple backticks (```) to define multi-line strings directly in the shell.

**In summary:**

* Use the Django shell for quick interactions and experimentation.
* Write your main application code in .py files within your project.
* Feel free to paste multiple lines of code into the terminal, especially in IPython.

Let me know if you have any more questions or want to move on to the next step of setting up database access!



ok, back to creating users, will these be thier actual emails, or thier emails insode my site? maining should i use thier actual email and will my appplication manage to send them an email or its just for users data sake?

Show drafts

Good question! The emails you provide when creating users in Django serve a dual purpose:

1. **User Identification:** Primarily, the email acts as a unique identifier for each user within your application. It's used for logging in, password resets, and other account-related actions.
2. **Potential for Communication:** If you intend to send emails from your application (e.g., welcome emails, password reset links, notifications), then using their actual emails is necessary. Django has built-in capabilities for sending emails, but you'll need to configure your email settings appropriately (we can discuss this later if needed).

**So, should you use their actual emails?**

* **If you plan to send emails:** Yes, definitely use their actual emails.
* **If you don't need to send emails:** You can still use their actual emails, or you can use placeholder emails (e.g., your\_username@example.com) if you prefer. Just keep in mind that these users won't be able to receive any emails from your application.

**Key points:**

* Django itself won't automatically send emails just because you create users with email addresses. You'll need to write the code to trigger and send those emails.
* Make sure you have the necessary permissions and comply with any relevant regulations (like GDPR) if you're collecting and using actual email addresses.

Let me know if you have any more questions or if you're ready to move on to setting up database access!



D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py shell

Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 8.20.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from django.contrib.auth.models import User

   ...:

   ...: # Create your user

   ...: you = User.objects.create\_user('omar', 'omar.essam@rowad-rme.

   ...: com', 'omar123')

   ...:

   ...: # Create Nada's user

   ...: nada = User.objects.create\_user('nada', 'nada.namir@rowad-rme

   ...: .com', 'nada123')

   ...:

   ...: # Create Yasser's user

   ...: yasser = User.objects.create\_user('yasser', 'yasser.abouzeid@

   ...: rowad-rme.com', 'yasser123')

----------------------------------------------------------------------OperationalError                     Traceback (most recent call last)File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:105, in CursorWrapper.\_execute(self, sql, params, \*ignored\_wrapper\_args)

    104 else:

--> 105     return self.cursor.execute(sql, params)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\sqlite3\base.py:354, in SQLiteCursorWrapper.execute(self, query, params)

    353 query = self.convert\_query(query, param\_names=param\_names)

--> 354 return super().execute(query, params)

OperationalError: no such table: auth\_user

The above exception was the direct cause of the following exception:

OperationalError                     Traceback (most recent call last)Cell In[1], line 4

      1 from django.contrib.auth.models import User

      3 # Create your user

----> 4 you = User.objects.create\_user('omar', 'omar.essam@rowad-rme.com', 'omar123')

      6 # Create Nada's user

      7 nada = User.objects.create\_user('nada', 'nada.namir@rowad-rme.com', 'nada123')

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\models.py:162, in UserManager.create\_user(self, username, email, password, \*\*extra\_fields)

    160 extra\_fields.setdefault("is\_staff", False)

    161 extra\_fields.setdefault("is\_superuser", False)

--> 162 return self.\_create\_user(username, email, password, \*\*extra\_fields)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\models.py:156, in UserManager.\_create\_user(self, username, email, password, \*\*extra\_fields)

    154 user = self.model(username=username, email=email, \*\*extra\_fields)

    155 user.password = make\_password(password)

--> 156 user.save(using=self.\_db)

    157 return user

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\base\_user.py:62, in AbstractBaseUser.save(self, \*args,

\*\*kwargs)

     61 def save(self, \*args, \*\*kwargs):

---> 62     super().save(\*args, \*\*kwargs)

     63     if self.\_password is not None:

     64         password\_validation.password\_changed(self.\_password, self)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py:891, in Model.save(self, force\_insert, force\_update, using, update\_fields, \*args)

    888     if loaded\_fields:

    889         update\_fields = frozenset(loaded\_fields)

--> 891 self.save\_base(

    892     using=using,

    893     force\_insert=force\_insert,

    894     force\_update=force\_update,

    895     update\_fields=update\_fields,

    896 )

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py:997, in Model.save\_base(self, raw, force\_insert, force\_update, using, update\_fields)

    993         force\_insert = self.\_validate\_force\_insert(force\_insert)

    994         parent\_inserted = self.\_save\_parents(

    995             cls, using, update\_fields, force\_insert

    996         )

--> 997     updated = self.\_save\_table(

    998         raw,

    999         cls,

   1000         force\_insert or parent\_inserted,

   1001         force\_update,

   1002         using,

   1003         update\_fields,

   1004     )

   1005 # Store the database on which the object was saved

   1006 self.\_state.db = using

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py:1160, in Model.\_save\_table(self, raw, cls, force\_insert, force\_update, using, update\_fields)

   1154 fields = [

   1155     f

   1156     for f in meta.local\_concrete\_fields

   1157     if not f.generated and (pk\_set or f is not meta.auto\_field)

   1158 ]

   1159 returning\_fields = meta.db\_returning\_fields

-> 1160 results = self.\_do\_insert(

   1161     cls.\_base\_manager, using, fields, returning\_fields, raw

   1162 )

   1163 if results:

   1164     for value, field in zip(results[0], returning\_fields):

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py:1201, in Model.\_do\_insert(self, manager, using, fields, returning\_fields, raw)

   1196 def \_do\_insert(self, manager, using, fields, returning\_fields, raw):

   1197     """

   1198     Do an INSERT. If returning\_fields is defined then this method should

   1199     return the newly created data for the model.

   1200     """

-> 1201     return manager.\_insert(

   1202         [self],

   1203         fields=fields,

   1204         returning\_fields=returning\_fields,

   1205         using=using,

   1206         raw=raw,

   1207     )

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\manager.py:87, in BaseManager.\_get\_queryset\_methods.<locals>.create\_method.<locals>.manager\_method(self, \*args, \*\*kwargs)

     85 @wraps(method)

     86 def manager\_method(self, \*args, \*\*kwargs):

---> 87     return getattr(self.get\_queryset(), name)(\*args, \*\*kwargs)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\query.py:1847, in QuerySet.\_insert(self, objs, fields, returning\_fields, raw, using, on\_conflict, update\_fields, unique\_fields)

   1840 query = sql.InsertQuery(

   1841     self.model,

   1842     on\_conflict=on\_conflict,

   1843     update\_fields=update\_fields,

   1844     unique\_fields=unique\_fields,

   1845 )

   1846 query.insert\_values(fields, objs, raw=raw)

-> 1847 return query.get\_compiler(using=using).execute\_sql(returning\_fields)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\sql\compiler.py:1836, in SQLInsertCompiler.execute\_sql(self, returning\_fields)

   1834 with self.connection.cursor() as cursor:

   1835     for sql, params in self.as\_sql():

-> 1836         cursor.execute(sql, params)

   1837     if not self.returning\_fields:

   1838         return []

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:122, in CursorDebugWrapper.execute(self, sql, params)

    120 def execute(self, sql, params=None):

    121     with self.debug\_sql(sql, params, use\_last\_executed\_query=True):

--> 122         return super().execute(sql, params)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:79, in CursorWrapper.execute(self, sql, params)     78 def execute(self, sql, params=None):

---> 79     return self.\_execute\_with\_wrappers(

     80         sql, params, many=False, executor=self.\_execute

     81     )

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:92, in CursorWrapper.\_execute\_with\_wrappers(self, sql, params, many, executor)

     90 for wrapper in reversed(self.db.execute\_wrappers):

     91     executor = functools.partial(wrapper, executor)

---> 92 return executor(sql, params, many, context)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:100, in CursorWrapper.\_execute(self, sql, params, \*ignored\_wrapper\_args)

     98     warnings.warn(self.APPS\_NOT\_READY\_WARNING\_MSG, category=RuntimeWarning)

     99 self.db.validate\_no\_broken\_transaction()

--> 100 with self.db.wrap\_database\_errors:

    101     if params is None:

    102         # params default might be backend specific.

    103         return self.cursor.execute(sql)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py:91, in DatabaseErrorWrapper.\_\_exit\_\_(self, exc\_type, exc\_value, traceback)

     89 if dj\_exc\_type not in (DataError, IntegrityError):

     90     self.wrapper.errors\_occurred = True

---> 91 raise dj\_exc\_value.with\_traceback(traceback) from exc\_value

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\utils.py:105, in CursorWrapper.\_execute(self, sql, params, \*ignored\_wrapper\_args)

    103     return self.cursor.execute(sql)

    104 else:

--> 105     return self.cursor.execute(sql, params)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\sqlite3\base.py:354, in SQLiteCursorWrapper.execute(self, query, params)

    352 param\_names = list(params) if isinstance(params, Mapping) else None

    353 query = self.convert\_query(query, param\_names=param\_names)

--> 354 return super().execute(query, params)

OperationalError: no such table: auth\_user

In [2]:

Show drafts

You encountered an OperationalError: no such table: auth\_user. This means Django is trying to interact with the auth\_user table in your database, but it doesn't exist yet.

**Why is this happening?**

Django's built-in user authentication system relies on a set of database tables to store user information, permissions, and other related data. These tables are created automatically when you run certain Django commands, but it seems you haven't executed those commands yet.

**How to fix it:**

1. **Make migrations:** Migrations are Django's way of keeping track of changes you make to your models (and the corresponding database tables). Run the following command to create the necessary migrations:

Bash

python manage.py makemigrations

Use code [with caution.](/faq#coding)

1. **Apply migrations:** This will actually create the tables in your database based on the migrations you just generated:

Bash

python manage.py migrate

Use code [with caution.](/faq#coding)

**After these steps:**

* The auth\_user table (and other related tables) should now exist in your database.
* You should be able to create users successfully using the code you provided earlier.

**Important note:** If you make any changes to your models (e.g., add new fields or change existing ones), you'll need to repeat the makemigrations and migrate steps to update your database schema accordingly.

Feel free to ask if you have any further questions or if you encounter any issues during the migration process. I'm here to help!



In [2]: python manage.py makemigrations

  Cell In[2], line 1

    python manage.py makemigrations

           ^

SyntaxError: invalid syntax , how do i exit back to be insode my djago project folder where manage.py is?

Show drafts

It seems you're still within the IPython shell, which is why you're getting a SyntaxError. To exit the shell and return to your Django project folder, simply type:

exit()

Press Enter, and you'll be back in your terminal at the location where you started the shell (your Django project folder).

Then, you can execute the makemigrations command correctly:

Bash

python manage.py makemigrations

Use code [with caution.](/faq#coding)

Feel free to ask if you have any more questions or need further assistance!



In [3]: exit()

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py makemigrations

No changes detected

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py migrate

Operations to perform:

  Apply all migrations: admin, auth, contenttypes, sessions

Running migrations:

  Applying contenttypes.0001\_initial... OK

  Applying auth.0001\_initial... OK

  Applying admin.0001\_initial... OK

  Applying admin.0002\_logentry\_remove\_auto\_add... OK

  Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK

  Applying contenttypes.0002\_remove\_content\_type\_name... OK

  Applying auth.0002\_alter\_permission\_name\_max\_length... OK

  Applying auth.0003\_alter\_user\_email\_max\_length... OK

  Applying auth.0004\_alter\_user\_username\_opts... OK

  Applying auth.0005\_alter\_user\_last\_login\_null... OK

  Applying auth.0006\_require\_contenttypes\_0002... OK

  Applying auth.0007\_alter\_validators\_add\_error\_messages... OK

  Applying auth.0008\_alter\_user\_username\_max\_length... OK

  Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK

  Applying auth.0010\_alter\_group\_name\_max\_length... OK

  Applying auth.0011\_update\_proxy\_permissions... OK

  Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK

  Applying sessions.0001\_initial... OK

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>

Show drafts

Great job! You successfully ran the makemigrations and migrate commands. The "No changes detected" message after makemigrations is normal since you haven't defined any custom models yet.

The migrate command output shows that Django applied several migrations related to its built-in apps (admin, auth, contenttypes, sessions). These migrations create the necessary database tables for user authentication, content management, and session handling.

Now that the required tables exist, you can proceed to create the users as we discussed earlier. Go ahead and open the Django shell again and execute the code to create the users for Omar, Nada, and Yasser.

Feel free to ask if you have any questions or need further guidance!



i went back to shell and pasted the create users code and this happened, i m not usure if ut worked, how do i check D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py shell

Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64

bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 8.20.0 -- An enhanced Interactive Python. Type '?' for help.

   ...:   Applying auth.0001\_initial... OK

   ...:   Applying admin.0001\_initial... OK

   ...:   Applying admin.0002\_logentry\_remove\_auto\_add... OK

   ...:   Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK

   ...:   Applying contenttypes.0002\_remove\_content\_type\_name... OK

   ...:   Applying auth.0002\_alter\_permission\_name\_max\_length... OK

   ...:   Applying auth.0003\_alter\_user\_email\_max\_length... OK

   ...:   Applying auth.0004\_alter\_user\_username\_opts... OK

   ...:   Applying auth.0005\_alter\_user\_last\_login\_null... OK

   ...:   Applying auth.0006\_require\_contenttypes\_0002... OK

   ...:   Applying auth.0007\_alter\_validators\_add\_error\_messages... OK

   ...:   Applying auth.0008\_alter\_user\_username\_max\_length... OK

   ...:   Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK

   ...:   Applying auth.0010\_alter\_group\_name\_max\_length... OK

   ...:   Applying auth.0011\_update\_proxy\_permissions... OK

   ...:   Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK

   ...:   Applying sessions.0001\_initial... OK

   ...:

   ...: D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\

   ...: 03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>

Show drafts

It seems there might be some confusion here. The output you're seeing indicates that you ran the migrate command again instead of pasting the code to create users within the Django shell.

Let's break down what happened and how to proceed:

1. **You exited the shell:** The exit() command successfully took you out of the Django shell and back to your terminal.
2. **You ran migrate again:** The subsequent output shows the result of running python manage.py migrate. This command applies any pending database migrations, but it doesn't create users.
3. **You need to re-enter the shell:** To create the users, you need to be within the Django shell.

Here's what you should do now:

1. **Start the shell:** Run the following command again to enter the Django shell:

Bash

python manage.py shell

Use code [with caution.](/faq#coding)

1. **Paste the code:** Once you're in the shell, paste the code to create the users (the code you provided in your previous message). Make sure you're pasting it directly into the shell, not into your terminal.
2. **Check if it worked:** After executing the code, you can verify if the users were created by running the following query in the shell:

Python

from django.contrib.auth.models import User

User.objects.all()

Use code [with caution.](/faq#coding)

This should display a list of all the users in your database, including the ones you just created.

**Remember:**

* The Django shell is an interactive environment where you can execute Python and Django code directly.
* Make sure you're pasting the code into the shell, not into your terminal.
* You can use queries like User.objects.all() within the shell to interact with your database and verify if the changes were successful.

Feel free to ask if you have any further questions or need more clarification! I'm here to help you through the process.



   ...: D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\

   ...: 03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.

   ...: py shell

  Cell In[1], line 3

    D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py makemigrations

       ^

SyntaxError: unexpected character after line continuation character

In [2]: from django.contrib.auth.models import User

   ...:

   ...: # Create your user

   ...: you = User.objects.create\_user('omar', 'omar.essam@rowad-rme.com',

   ...:  'omar123')

   ...:

   ...: # Create Nada's user

   ...: nada = User.objects.create\_user('nada', 'nada.namir@rowad-rme.com'

   ...: , 'nada123')

   ...:

   ...: # Create Yasser's user

   ...: yasser = User.objects.create\_user('yasser', 'yasser.abouzeid@rowad

   ...: -rme.com', 'yasser123')

In [3]: from django.contrib.auth.models import User

   ...: User.objects.all()

Out[3]: <QuerySet [<User: omar>, <User: nada>, <User: yasser>]>

In [4]:  , great

Show drafts

Fantastic! You've successfully created the users, and you've even verified their existence in the database using the User.objects.all() query. The output <QuerySet [<User: omar>, <User: nada>, <User: yasser>]> confirms that the users were created and are now stored in your database.

You're making excellent progress! Now that you have your users set up, we can move on to the next crucial step: configuring Django to access your Microsoft SQL database.

Are you ready to proceed with that?



yes ready, here is a python code thats already running and it connects to the database whenever we re on network, import pyodbc

def connect\_to\_sql\_server(server, database, user, password):

    """Connects to a SQL Server database and returns a connection object."""

    conn\_str = (

        r'DRIVER={SQL Server};'

        r'SERVER=' + server + ';'

        r'DATABASE=' + database + ';'

        r'UID=' + user + ';'

        r'PWD=' + password + ';'

    )

    conn = pyodbc.connect(conn\_str)

    return conn

def execute\_query(conn, query):

    """Executes a SQL query and returns the results as a list of tuples."""

    cursor = conn.cursor()

    cursor.execute(query)

    results = cursor.fetchall()

    cursor.close()

    return results

if \_\_name\_\_ == "\_\_main\_\_":

    server = '10.10.11.241'

    database = 'omar.rme1'

    user = 'yasser'

    password = 'yasser123'

    query = """

        SELECT project\_no, SUM(amount) AS total\_cost

        FROM cost\_dist

        WHERE project\_no IN ('122', '130', '143', '152', '157', '166','167', '168', '169')

        GROUP BY project\_no;

    """

    try:

        conn = connect\_to\_sql\_server(server, database, user, password)

        results = execute\_query(conn, query)

        for row in results:

            print(row)

    except Exception as e:

        print("Error:", e)

    finally:

        if conn:

            conn.close() , now in my web application the users will change the project no in that query and the output to them will be the amount (from that same table cost\_dist) now we need to make the site also availabe locally on network, i have access to the server that contains the microsoft sql database, we can host it there and they can connect from chrome, correct?

Show drafts

Yes, you're absolutely on the right track! You can definitely host your Django web application on the same server that contains your Microsoft SQL database. This would allow users on the network to access the application through their web browsers (like Chrome) by simply navigating to the appropriate URL.

Here's a breakdown of how this would work:

1. **Database Configuration:** You'll need to configure your Django project to connect to the Microsoft SQL database on the same server. You'll essentially provide the same connection details (server, database, user, password) that you're using in your existing Python code to Django's database settings.
2. **Model Creation:** You'll create Django models to represent the relevant tables in your database (at least the cost\_dist table in this case). This will allow you to interact with the data through Django's ORM.
3. **View and Form Creation:** You'll create a Django view to handle user requests (project number input) and a form to collect that input.
4. **Query Execution:** Within the view, you'll use Django's ORM to execute a query similar to the one in your Python code, but with the project number(s) provided by the user through the form.
5. **Template Rendering:** You'll create a template to display the query results (project number and total cost) in a user-friendly format.
6. **Deployment on the Server:** You'll deploy your Django project on the server. This typically involves setting up a web server (like Apache or Nginx) and configuring it to serve your Django application.

Once deployed, users on the network can access your application by entering the server's IP address or hostname (along with the appropriate port and path) in their web browsers. They can then input project numbers, and your application will fetch and display the corresponding total costs from the database.

Let's start by configuring your Django project to connect to the Microsoft SQL database. Are you ready to proceed with that step?



yes, will we write the code as if we re on that server where the database is on local without the need for an ip ( i can remote desktop connection to it if we need to ) or will we write code and work from my laptop where i can query the database but cant host the site?

Show drafts

Excellent question, and it's an important distinction! Here's how we'll handle the database connection configuration:

**We'll write the code as if we're on the server.**

This means that when you configure Django's database settings, you'll use:

* 'localhost' or '127.0.0.1' as the HOST (since the database is on the same server where you'll be hosting the Django application).
* The actual database name, username, and password you provided earlier.

**Development workflow:**

* You'll develop and test the Django application on your laptop.
* During development, you can configure your laptop's Django settings to connect to the remote database using its IP address (as you're doing in your existing Python code).
* Once you're ready to deploy, you'll update the Django settings on the server to use localhost as the HOST.

**Advantages of this approach:**

* **Simpler configuration on the server:** The final configuration on the server will be straightforward, as it only needs to connect to a local database.
* **Easier testing:** You can easily test the database connection and interactions on your laptop during development.
* **Flexibility:** If you ever need to move the database or the application to a different server, you'll only need to update the HOST setting in Django's configuration.

**Key point:** Make sure you have the necessary network connectivity and permissions to access the database from your laptop during development.

Let me know if you have any other questions or if you're ready to start configuring the database connection in your Django project!



i am ready, will be a py file like the ones i make with pyodbc or will we change things in django config files that i dont know about?

Show drafts

[github.com/Snowflake-Labs/sfquickstarts](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

subject to license (Apache - 2.0)

We'll be changing things in Django's configuration files. Specifically, you'll modify the DATABASES setting in your project's settings.py file. This is where Django stores all the information it needs to connect to your database.

Here's a breakdown of what we'll do:

1. **Locate settings.py:**
   * This file is located within the omar\_warehouse folder (the inner one, not the outer one with the same name).
   * The path should be something like: omar\_warehouse/omar\_warehouse/settings.py
2. **Open settings.py in your code editor:**
   * Use your preferred code editor (VS Code, PyCharm, etc.) to open this file.
3. **Find the DATABASES setting:**
   * It should look something like this:

Python

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

Use code [with caution.](/faq#coding)

* + This default configuration is for a SQLite database, which is not what you're using. We'll need to change this.

[1. github.com](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

[Apache - 2.0](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

[github.com](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

1. **Install the necessary database backend:**
   * Django needs a specific backend to interact with Microsoft SQL Server. You'll need to install it using pip:

Bash

pip install django-mssql-backend

Use code [with caution.](/faq#coding)

* + - Make sure you're in your activated virtual environment when you run this command.

1. **Modify the DATABASES setting:**
   * Change the DATABASES setting to look like this:

Python

DATABASES = {

'default': {

'ENGINE': 'mssql', # Use the django-mssql-backend

'NAME': 'omar.rme1', # Your database name

'USER': 'yasser', # Your database username

'PASSWORD': 'yasser123', # Your database password

'HOST': 'localhost', # Since the database is on the same server

'PORT': '', # Default SQL Server port (leave blank)

'OPTIONS': {

'driver': 'ODBC Driver 17 for SQL Server', # Adjust if needed

},

}

}

Use code [with caution.](/faq#coding)

* + Make sure you replace the NAME, USER, and PASSWORD with your actual database credentials.
  + The driver option might need to be adjusted depending on the specific ODBC driver you have installed on the server. You can usually find the correct driver name in the ODBC Data Source Administrator on the server.

**After these changes:**

* Django should now be configured to connect to your Microsoft SQL database on the server.
* You can proceed to create Django models to represent your database tables.

Feel free to ask if you have any questions or need help with any of these steps!

Sources and related content

[github.com](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

[Apache - 2.0](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)

[github.com](https://github.com/Snowflake-Labs/sfquickstarts" \t "_blank)



i will use these settings later when I move my project to the server, but for now on my laptop, i used these settings: DATABASES = {

    'default': {

        'ENGINE': 'mssql',  # Use the django-mssql-backend

        'NAME': 'omar.rme1',  # Your database name

        'USER': 'yasser',     # Your database username

        'PASSWORD': 'yasser123', # Your database password

        'HOST': '10.10.11.241',   # When I'm on My Laptop

        'PORT': '1433',           # Default SQL Server port 1433

        'OPTIONS': {

            'driver': 'ODBC Driver 17 for SQL Server',  # Adjust if needed

        },

    }

} , correct?

Show drafts

Yes, those settings look correct for connecting to your Microsoft SQL database from your laptop during development!

Here's a breakdown of the key points:

* \*\*ENGINE: 'mssql':\*\* This tells Django to use thedjango-mssql-backend` you installed earlier to interact with the database.
* **NAME, USER, PASSWORD:** These are your actual database credentials.
* **HOST: '10.10.11.241'`:** This is the IP address of the server where your database is located. It allows you to connect to the database remotely from your laptop.
* **PORT: '1433'`:** This is the default port for SQL Server. If your database is using a different port, you'll need to adjust this accordingly.
* **OPTIONS: {'driver': 'ODBC Driver 17 for SQL Server'}`:** This specifies the ODBC driver to use for the connection. Make sure this matches the driver you have installed on your laptop.

**Important:**

* **Remember to change HOST to 'localhost' when you deploy your project to the server.** This is because, on the server, the database will be local, and you won't need to specify its IP address.
* **Ensure network connectivity:** Make sure your laptop can reach the server at the specified IP address and port. You might need to configure your network or firewall settings if you encounter any connection issues.

With these settings in place, Django should be able to establish a connection to your Microsoft SQL database. You can now proceed to create Django models to represent your database tables, starting with the cost\_dist table.

Feel free to ask if you have any more questions or need further assistance!



ok great, cn we have any kind of test now to see if django can connect to the server mssql?

Show drafts

Yes, absolutely! We can perform a simple test to check if Django can successfully connect to your Microsoft SQL Server database.

Here's how you can do it:

1. **Open the Django shell:**
   * Make sure you're in your activated virtual environment.
   * Run the following command in your terminal:

Bash

python manage.py shell

Use code [with caution.](/faq#coding)

1. **Import the connection object:**
   * Once you're in the shell, import the connection object from Django's database module:

Python

from django.db import connection

Use code [with caution.](/faq#coding)

1. **Check the connection:**
   * Use the connection.ensure\_connection() method to attempt to establish a connection to the database:

Python

connection.ensure\_connection()

Use code [with caution.](/faq#coding)

* + If the connection is successful, you won't see any output or errors.
  + If there's an issue with the connection (e.g., incorrect credentials, network problems, or driver issues), you'll get an exception with a descriptive error message.

**Example of a successful connection:**

In [1]: from django.db import connection

In [2]: connection.ensure\_connection()

In [3]:

**Example of a failed connection (incorrect password):**

In [1]: from django.db import connection

In [2]: connection.ensure\_connection()

---------------------------------------------------------------------------

OperationalError Traceback (most recent call last)

Cell In[2], line 1

----> 1 connection.ensure\_connection()

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\base\base.py:322, in BaseDatabaseWrapper.ensure\_connection(self)

320 with self.\_lock:

321 if self.connection is None:

--> 322 self.connect()

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\base\base.py:283, in BaseDatabaseWrapper.connect(self)

281 return

282 # Check for invalid configurations.

--> 283 self.\_validate\_connection\_config()

284 # In case the previous \_close() call has set self.connection to None,

285 # pass the clean-up reference to the new connection.

286 conn = self.connection = self.\_cursor()

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\backends\base\base.py:385, in BaseDatabaseWrapper.\_validate\_connection\_config(self)

383 raise ImproperlyConfigured(

384 f"settings.DATABASES is improperly configured. Please supply the {missing} for database "

--> 385 f"'{self.alias}'."

386 )

ImproperlyConfigured: settings.DATABASES is improperly configured. Please supply the ENGINE value for database 'default'.

**Interpretation:**

* **No output:** If you don't see any output or errors after running connection.ensure\_connection(), it means Django was able to connect to your database successfully.
* **Error message:** If you get an error, carefully read the message. It will usually provide clues about what went wrong (e.g., incorrect credentials, network issues, driver problems).

Go ahead and try this test. Let me know the outcome, and we can proceed from there!



you keep saying make sure you re in your virtual env , why is that? i already pip installed django and mssql globaly, what will go wrong if i edite or worked in my project and went into shell without going into a venv first?

Show drafts

You raise a crucial point about virtual environments! While you've installed Django and the MSSQL backend globally, it's highly recommended to work within a virtual environment for several important reasons:

**1. Dependency Isolation:**

* **Project-specific dependencies:** Each Django project might require different versions of Django or other third-party libraries. Virtual environments ensure that each project has its own isolated set of dependencies, preventing conflicts between different projects.
* **Example:** Imagine you have two Django projects: one uses Django 3.2 and another uses Django 4.0. If you install both versions globally, they might clash and cause unexpected behavior. Virtual environments prevent this by keeping the dependencies for each project separate.

**2. Clean Project Structure:**

* **Avoid cluttering your global Python installation:** Installing all project dependencies globally can lead to a messy and difficult-to-manage Python environment.
* **Example:** Over time, you might accumulate numerous libraries from various projects, making it challenging to track which libraries belong to which project and potentially leading to version conflicts.

**3. Reproducibility:**

* **Consistent development and deployment:** Virtual environments help ensure that your project runs consistently across different development machines and production servers. By specifying the exact dependencies in a requirements.txt file within your virtual environment, you can easily recreate the same environment elsewhere.

**What can go wrong without a virtual environment:**

* **Version conflicts:** You might encounter errors due to incompatible library versions if you install everything globally.
* **Difficulty debugging:** It can be challenging to pinpoint the source of problems if multiple projects share the same global dependencies.
* **Deployment issues:** Your project might not work as expected on a production server if the environment is different from your development environment.

**In summary:**

While it's technically possible to work without virtual environments, it's strongly discouraged, especially for Django projects. They provide essential benefits like dependency isolation, clean project structure, and reproducibility, making your development process smoother and more reliable.

I highly recommend activating your virtual environment before editing your project files or entering the Django shell. This ensures that you're working with the correct set of dependencies and minimizes the risk of encountering unexpected issues.

Let me know if you have any further questions or would like assistance with activating your virtual environment!



ok you convinced me , not right now, but i will keep in mind to set up a venv from now on and make a requiremnts txt, now i will scroll up and follow the steps you told me and get back to you

Show drafts

Excellent! That's a wise decision. Virtual environments are a valuable tool for any Python project, and they'll definitely make your Django development experience smoother.

I'm here whenever you're ready to create a virtual environment and set up your requirements.txt. Just let me know, and I'll guide you through the process.

For now, go ahead and follow the steps we discussed earlier to test the database connection. If you encounter any issues or have further questions, don't hesitate to ask! I'll be happy to assist you.



great In [4]: from django.db import connection

In [5]: connection.ensure\_connection()

In [6]:

Show drafts

Perfect! It looks like the connection was successful. Since you didn't receive any errors after running connection.ensure\_connection(), Django was able to establish a connection to your Microsoft SQL Server database using the settings you provided.

This is a great milestone! Now you can confidently move on to the next step: creating Django models to represent your database tables.

Are you ready to start defining your models?



i dont know what that exactly means, so i ll explain the cost\_dist table to you, it has the data for the cost on all the company's projects, including qty and unit rate if there is any, type of cost and expenditure type, here are the column headers and you ll understand what it entails :

[Opens in a new window](https://lens.google.com/search?dp=CtECGs4CaHR0cHM6Ly9saDMuZ29vZ2xldXNlcmNvbnRlbnQuY29tL2dnL0FKSXZYaXVlQ1dzSEZqTGpVY1JLNm1CQzN3Qkl1QkZaYmR4ZnZybGkzalZsX3lHLWRGR1YyeDRFMUhQLWhLLWtFU0R5Q1BNdmUwYl95UnhPM1EzZHhKdnduWEJsbzEyZDVVSDBxSHdCczJfazBUY0NaN3huei00dVhzeHoyYlFkVzJHUHY5NkYyWU1oYWJjcVFyRVZ3cmltakY4dkZOUlhKbnkyVi0wcVBYM25JbHNkWWNjeHNhZGtKcXdzVFRrU0N1QjhOX291TXZvYjVzUjhUNUtxTFlkWEctZzI5a2JTOFRESWI2SERnck1FNHNBejlGelVNX1RON0UyR29UbHQ2bUZLeTVhaXQ0R01Td04tenk0bVhIamthUTRYQXRBcTFKMTc0S1U9ZDIFCCMQ7CM=&ep=bia)

Show drafts

Certainly, let's break down the cost\_dist table based on the column headers you provided:

**Column Headers and Interpretation**

* **cost\_dist\_id:** A unique identifier for each row in the table (likely an auto-incrementing integer).
* **project\_no:** The project number associated with the cost entry.
* **acc\_no:** The account number related to the cost (could be a general ledger account or a project-specific account).
* **date:** The date the cost was incurred.
* **doc\_no:** The document number associated with the cost (e.g., invoice number, receipt number).
* **desc:** A description of the cost.
* **amount:** The monetary value of the cost.
* **qty:** The quantity of items or services associated with the cost (if applicable).
* **unit\_rate:** The unit price of the item or service (if applicable).
* **type:** The type of cost (e.g., labor, material, overhead).
* **exp\_type:** The expenditure type (e.g., direct cost, indirect cost).

**Understanding the Data**

From these columns, it appears that the cost\_dist table stores detailed information about the costs incurred on various projects within the company. Each row represents a specific cost entry, linked to a project, account, and potentially a document.

The amount, qty, and unit\_rate columns allow for tracking both the total cost and the quantity/unit price breakdown, which can be useful for analysis.

The type and exp\_type columns provide categorization for further analysis and reporting on different cost types and expenditure categories.

**Next Steps**

With this understanding of the table structure, we can now proceed to create a Django model that accurately represents this data. We'll define fields in the model that correspond to these columns, specifying their data types and any additional constraints or relationships.

Feel free to ask if you have any more questions or if you're ready to move on to creating the Django model.



i will need only what the users need, so first are project\_no and project\_name ( which will be their input ) next is gl\_date where they may also select a range of time where they want the data and finally the output is line\_desc (description of the item) and unit , qty and amount , i want 2 options they can select either project\_no or project\_name (so we need to get a unique list of those so that we can have a dropdown list on the site) then 1 option to see the total amount of the project and the other option is to see a detailed amount of the project with all the lines (maybe a download excel option if possible)

Show drafts

OK, let's outline the key elements of your Django model and the desired functionality based on your requirements.

**Model Fields:**

* **project\_no:** (CharField) - To store the project number (user input). You'll likely want to set unique=True to ensure project numbers are unique.
* **project\_name:** (CharField) - To store the project name (user input).
* **gl\_date:** (DateField) - To store the date associated with each cost entry.
* **line\_desc:** (CharField) - To store the description of the item or service.
* **unit:** (CharField) - To store the unit of measurement for the item or service.
* **qty:** (DecimalField or IntegerField) - To store the quantity.
* **amount:** (DecimalField) - To store the monetary amount.

**Functionality:**

1. **User Input:**
   * Dropdown list for selecting project\_no or project\_name.
   * Date range selection for filtering by gl\_date.
2. **Query and Output:**
   * **Option 1: Total Amount:**
     + Retrieve the total amount for the selected project (filtered by date range if applicable).
     + Display the total amount to the user.
   * **Option 2: Detailed List:**
     + Retrieve all rows ( line\_desc, unit, qty, amount) for the selected project (filtered by date range if applicable).
     + Display the detailed list to the user.
     + (Optional) Provide a download option to export the data as an Excel file.

**Additional Considerations:**

* **Dropdown Data:** You'll need to populate the dropdown lists with unique project\_no and project\_name values from your database. You can achieve this by querying your database and extracting the distinct values.
* **Excel Export:** If you want to implement the Excel export feature, you'll likely need to use a third-party library like openpyxl or xlsxwriter.

**Next Steps:**

1. **Create the Django model:** Define the model in your models.py file with the fields mentioned above.
2. **Generate and apply migrations:** Run python manage.py makemigrations and python manage.py migrate to create the corresponding database table.
3. **Build views and forms:** Create views to handle user input and queries, and forms to collect the input data.
4. **Design templates:** Create templates to display the input form, total amount, and detailed list (if applicable).
5. **Implement Excel export (optional):** If needed, add the functionality to export the detailed list as an Excel file.

I'm ready to assist you with each of these steps. Feel free to ask any questions or request further guidance as we proceed.



you understood me correctly in our goals, thank you, lets define the fields in models.py

Show drafts

Let's get started with defining the fields in your models.py file.

**Navigate to models.py**

1. Open your project's models.py file. It should be located at: omar\_warehouse/omar\_warehouse/models.py
2. Add the following code to define your CostDist model:

Python

from django.db import models

class CostDist(models.Model):

project\_no = models.CharField(max\_length=50, unique=True)

project\_name = models.CharField(max\_length=200)

gl\_date = models.DateField()

line\_desc = models.CharField(max\_length=200)

unit = models.CharField(max\_length=50)

qty = models.DecimalField(max\_digits=10, decimal\_places=2) # Adjust max\_digits and decimal\_places as needed

amount = models.DecimalField(max\_digits=15, decimal\_places=2) # Adjust max\_digits and decimal\_places as needed

def \_\_str\_\_(self):

return f"{self.project\_no} - {self.project\_name}"

Use code [with caution.](/faq#coding)

**Explanation of the code:**

* We import the necessary models module from Django.
* We create a class named CostDist that inherits from models.Model. This tells Django that this class represents a database table.
* Inside the class, we define fields corresponding to the columns you mentioned:
  + project\_no is a CharField with a maximum length of 50 characters. We set unique=True to ensure that each project number is unique in the database.
  + project\_name is also a CharField with a maximum length of 200 characters.
  + gl\_date is a DateField to store dates.
  + line\_desc, unit are CharField fields for descriptions and units.
  + qty and amount are DecimalField to store numerical values with decimal precision. You can adjust max\_digits and decimal\_places based on the expected range and precision of your data.
* The \_\_str\_\_ method provides a human-readable representation of a CostDist object, which is helpful when you're interacting with these objects in the Django admin or shell.

**Next steps:**

1. **Make migrations:** Run the following command in your terminal to create the migration files:

Bash

python manage.py makemigrations

Use code [with caution.](/faq#coding)

1. **Apply migrations:** Run the following command to create the actual table in your database:

Bash

python manage.py migrate

Use code [with caution.](/faq#coding)

Once you've completed these steps, you'll have a cost\_dist table in your database, ready to store data. We can then move on to populating the dropdown lists and building the views and forms for your web application.

Feel free to ask if you have any questions or need further clarification along the way.



there is no models.py file inside D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\omar\_warehouse

Show drafts

You're absolutely correct. I apologize for the oversight. In a new Django project, the models.py file is indeed located within the app directory, not directly in the project's root directory.

**Here's the corrected path:**

* omar\_warehouse/omar\_warehouse/ <app\_name> /models.py

**Key point:**

* Django projects are typically organized into multiple apps. Each app represents a specific functionality or feature of your web application.
* By default, Django creates a basic app named omar\_warehouse when you start a new project.

**Next Steps:**

1. **Identify the app name:**

* Check the INSTALLED\_APPS setting in your omar\_warehouse/omar\_warehouse/settings.py file. It should list the apps included in your project.
* The default app is usually named the same as your project, so in this case, it's likely 'omar\_warehouse'.

1. **Navigate to the correct models.py:**

* Once you know the app name, open the models.py file within that app's directory.
* The path should be: omar\_warehouse/omar\_warehouse/ <app\_name> /models.py

1. **Define your model:**

* Paste the code we discussed earlier into this models.py file to define your CostDist model.

1. **Make and apply migrations:**

* Run the makemigrations and migrate commands as explained before to create the corresponding database table.

I apologize for any confusion caused by the incorrect path. Please let me know if you have any further questions or need assistance with any of these steps!



there are the installed apps in setting # Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

] but there is no app folders inside D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\omar\_warehouse

Show drafts

You are absolutely right. In a newly created Django project, there wouldn't be any app folders by default within the omar\_warehouse/omar\_warehouse directory. Django encourages a modular structure where you organize your project into different apps, but you need to create these apps explicitly.

**Let's create an app for your CostDist model:**

1. **Navigate to your project's root directory:**

* Open your terminal or command prompt and navigate to D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse

1. **Create a new app:**

* Use the following command to create a new app named cost\_management (you can choose any suitable name):

Bash

python manage.py startapp cost\_management

Use code [with caution.](/faq#coding)

* This will create a new folder named cost\_management within your omar\_warehouse/omar\_warehouse directory.

1. **Add the app to INSTALLED\_APPS:**

* Open your omar\_warehouse/omar\_warehouse/settings.py file.
* Locate the INSTALLED\_APPS setting.
* Add 'cost\_management' to the list:

Python

INSTALLED\_APPS = [

# ... other default apps ...

'cost\_management', # Add your new app here

]

Use code [with caution.](/faq#coding)

1. **Define your model:**

* Open the models.py file within your newly created app (omar\_warehouse/omar\_warehouse/cost\_management/models.py).
* Paste the code we discussed earlier to define your CostDist model in this file.

1. **Make and apply migrations:**

* Run the makemigrations and migrate commands as explained before to create the corresponding database table.

Now you have a dedicated app (cost\_management) to manage your cost-related data, and your CostDist model is defined within that app.

Feel free to ask if you have any further questions or need more guidance as we move forward!



D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py startapp cost\_dist

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse> , the folder cost\_dist was created in D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse not in D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\omar\_warehouse , is that correct, should i continue?

Show drafts

Yes, that is correct! When you create a new app using the startapp command, Django places the app folder directly within the project's main directory (omar\_warehouse in your case).

It's not necessary for the app folder to be inside the inner omar\_warehouse directory. Django's configuration and structure allow it to recognize and work with apps located in the project's main directory.

So, you can definitely continue with the cost\_dist app you just created. Just make sure to add it to the INSTALLED\_APPS list in your omar\_warehouse/omar\_warehouse/settings.py file as we discussed earlier.

Feel free to proceed with defining your CostDist model within the cost\_dist/models.py file and then run the migrations to create the corresponding database table.

If you have any more questions or concerns along the way, don't hesitate to ask!



INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'cost\_dist',

]

added, will now go to models.py

Show drafts

Great! You've successfully added your cost\_dist app to the INSTALLED\_APPS list in your settings. This is an essential step to let Django know about your new app and include its functionality in your project.

Now, go ahead and navigate to the models.py file within your cost\_dist app (omar\_warehouse/cost\_dist/models.py) and define your CostDist model as we discussed earlier. Once you've defined the model, remember to run the makemigrations and migrate commands to create the corresponding table in your database.

Feel free to ask if you have any questions or need further guidance along the way!



D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py startapp cost\_dist

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py makemigrations

Traceback (most recent call last):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 113, in load\_backend

    return import\_module("%s.base" % backend\_name)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1310, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1324, in \_find\_and\_load\_unlocked

ModuleNotFoundError: No module named 'mssql'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 22, in

<module>

    main()

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 18, in

main

    execute\_from\_command\_line(sys.argv)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 442, in execute\_from\_command\_line

    utility.execute()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 416, in execute

    django.setup()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\\_\_init\_\_.py", line 24, in setup

    apps.populate(settings.INSTALLED\_APPS)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\registry.py", line 116, in populate

    app\_config.import\_models()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\config.py", line 269, in import\_models

    self.models\_module = import\_module(models\_module\_name)

                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1331, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 935, in \_load\_unlocked

  File "<frozen importlib.\_bootstrap\_external>", line 995, in exec\_module

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\models.py", line 5, in <module>

    from django.contrib.auth.base\_user import AbstractBaseUser, BaseUserManager

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\base\_user.py", line 40, in <module>

    class AbstractBaseUser(models.Model):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 143, in \_\_new\_\_

    new\_class.add\_to\_class("\_meta", Options(meta, app\_label))

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 371, in add\_to\_class

    value.contribute\_to\_class(cls, name)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\options.py", line 231, in contribute\_to\_class

    self.db\_table, connection.ops.max\_name\_length()

                   ^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 15, in \_\_getattr\_\_

    return getattr(self.\_connections[self.\_alias], item)

                   ~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 62, in \_\_getitem\_\_

    conn = self.create\_connection(alias)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 193, in create\_connection

    backend = load\_backend(db["ENGINE"])

              ^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 126, in load\_backend

    raise ImproperlyConfigured(

django.core.exceptions.ImproperlyConfigured: 'mssql' isn't an available database backend or couldn't be imported. Check the above exception. To use one of the built-in backends, use 'django.db.backends.XXX', where XXX is one

of:

    'mysql', 'oracle', 'postgresql', 'sqlite3'

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse> , i will try to pip install the needed modules

Show drafts

Yes, it seems you're encountering an error because the mssql database backend is not available in your Django project. This is likely because you haven't installed the required package yet, even though you mentioned installing it globally.

**Remember:** Even if you've installed a package globally, you still need to install it within your virtual environment for Django to recognize and use it.

**Here's what you need to do:**

1. **Activate your virtual environment:** Make sure you're in the activated virtual environment for your omar\_warehouse project.
2. **Install django-mssql-backend:** Run the following command in your terminal (while the virtual environment is activated):

Bash

pip install django-mssql-backend

Use code [with caution.](/faq#coding)

1. **Retry the makemigrations command:** Once the installation is complete, try running python manage.py makemigrations again. This time, Django should be able to find the mssql backend and proceed with creating the migrations for your CostDist model.

**Explanation:**

* Virtual environments keep project dependencies isolated. Even if you have django-mssql-backend installed globally, Django won't be able to access it unless it's also installed within the virtual environment where your project is running.
* The makemigrations command needs to interact with your database backend to generate the migration files. Without the mssql backend available in the virtual environment, it can't complete this process.

**Key takeaway:** Always install your project dependencies (including database backends) within your activated virtual environment to ensure that Django can find and use them correctly.

Let me know if you have any further questions or encounter any issues during the installation or migration process. I'm here to help!



D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py makemigrations

Traceback (most recent call last):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 113, in load\_backend

    return import\_module("%s.base" % backend\_name)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1310, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1324, in \_find\_and\_load\_unlocked

ModuleNotFoundError: No module named 'mssql'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 22, in

<module>

    main()

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 18, in

main

    execute\_from\_command\_line(sys.argv)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 442, in execute\_from\_command\_line

    utility.execute()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 416, in execute

    django.setup()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\\_\_init\_\_.py", line 24, in setup

    apps.populate(settings.INSTALLED\_APPS)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\registry.py", line 116, in populate

    app\_config.import\_models()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\config.py", line 269, in import\_models

    self.models\_module = import\_module(models\_module\_name)

                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1331, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 935, in \_load\_unlocked

  File "<frozen importlib.\_bootstrap\_external>", line 995, in exec\_module

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\models.py", line 5, in <module>

    from django.contrib.auth.base\_user import AbstractBaseUser, BaseUserManager

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\base\_user.py", line 40, in <module>

    class AbstractBaseUser(models.Model):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 143, in \_\_new\_\_

    new\_class.add\_to\_class("\_meta", Options(meta, app\_label))

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 371, in add\_to\_class

    value.contribute\_to\_class(cls, name)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\options.py", line 231, in contribute\_to\_class

    self.db\_table, connection.ops.max\_name\_length()

                   ^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 15, in \_\_getattr\_\_

    return getattr(self.\_connections[self.\_alias], item)

                   ~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 62, in \_\_getitem\_\_

    conn = self.create\_connection(alias)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 193, in create\_connection

    backend = load\_backend(db["ENGINE"])

              ^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 126, in load\_backend

    raise ImproperlyConfigured(

django.core.exceptions.ImproperlyConfigured: 'mssql' isn't an available database backend or couldn't be imported. Check the above exception. To use one of the built-in backends, use 'django.db.backends.XXX', where XXX is one

of:

    'mysql', 'oracle', 'postgresql', 'sqlite3'

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse>python manage.py makemigrations

Traceback (most recent call last):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 113, in load\_backend

    return import\_module("%s.base" % backend\_name)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1310, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1324, in \_find\_and\_load\_unlocked

ModuleNotFoundError: No module named 'mssql'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 22, in

<module>

    main()

  File "D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse\manage.py", line 18, in

main

    execute\_from\_command\_line(sys.argv)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 442, in execute\_from\_command\_line

    utility.execute()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\core\management\\_\_init\_\_.py", line 416, in execute

    django.setup()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\\_\_init\_\_.py", line 24, in setup

    apps.populate(settings.INSTALLED\_APPS)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\registry.py", line 116, in populate

    app\_config.import\_models()

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\apps\config.py", line 269, in import\_models

    self.models\_module = import\_module(models\_module\_name)

                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\importlib\\_\_init\_\_.py", line 90, in import\_module

    return \_bootstrap.\_gcd\_import(name[level:], package, level)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import

  File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load

  File "<frozen importlib.\_bootstrap>", line 1331, in \_find\_and\_load\_unlocked

  File "<frozen importlib.\_bootstrap>", line 935, in \_load\_unlocked

  File "<frozen importlib.\_bootstrap\_external>", line 995, in exec\_module

  File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\models.py", line 5, in <module>

    from django.contrib.auth.base\_user import AbstractBaseUser, BaseUserManager

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\contrib\auth\base\_user.py", line 40, in <module>

    class AbstractBaseUser(models.Model):

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 143, in \_\_new\_\_

    new\_class.add\_to\_class("\_meta", Options(meta, app\_label))

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\base.py", line 371, in add\_to\_class

    value.contribute\_to\_class(cls, name)

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\models\options.py", line 231, in contribute\_to\_class

    self.db\_table, connection.ops.max\_name\_length()

                   ^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 15, in \_\_getattr\_\_

    return getattr(self.\_connections[self.\_alias], item)

                   ~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\utils\connection.py", line 62, in \_\_getitem\_\_

    conn = self.create\_connection(alias)

           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 193, in create\_connection

    backend = load\_backend(db["ENGINE"])

              ^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "C:\Users\Omar Essam\AppData\Local\Programs\Python\Python312\Lib\site-packages\django\db\utils.py", line 126, in load\_backend

    raise ImproperlyConfigured(

django.core.exceptions.ImproperlyConfigured: 'mssql' isn't an available database backend or couldn't be imported. Check the above exception. To use one of the built-in backends, use 'django.db.backends.XXX', where XXX is one

of:

    'mysql', 'oracle', 'postgresql', 'sqlite3'

D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse> , i still got an error after i pip installed, i will create a new venu and try again , what are all the libraries that we already used or we will need so that i make a requirements.txt?

Show drafts

[github.com/MATH-517/math\_517\_website](https://github.com/MATH-517/math_517_website" \t "_blank)

You're absolutely right. Creating a new virtual environment and installing the required packages there is a great way to ensure a clean and isolated environment for your Django project.

Here are the libraries we've used or will likely need for your project, which you can include in your requirements.txt file:

* **Django:** The core Django framework.
* **django-mssql-backend:** The database backend for connecting to Microsoft SQL Server.
* **pyodbc:** (Likely needed) For interacting with the database using ODBC. Although Django's ORM handles most database interactions, you might still need pyodbc for certain low-level operations or if you have existing code that uses it.
* **openpyxl or xlsxwriter:** (Optional) If you want to implement the Excel export feature.

**Creating the virtual environment and requirements.txt**

1. **Navigate to your project's root directory:**
   * Open your terminal and go to D:\OneDrive\OneDrive - Rowad Modern Engineering\x004 Data Science\03.rme.db\00.repo\rme.db-1\13.django\omar\_warehouse.
2. **Create a new virtual environment:**
   * Run the following command:

Bash

python -m venv my\_project\_env # Replace 'my\_project\_env' with your desired name

Use code [with caution.](/faq#coding)

1. **Activate the virtual environment:**
   * On Windows:

Bash

my\_project\_env\Scripts\activate

Use code [with caution.](/faq#coding)

* + On macOS/Linux:

Bash

source my\_project\_env/bin/activate

[1. github.com](https://github.com/MATH-517/math_517_website" \t "_blank)

[github.com](https://github.com/MATH-517/math_517_website" \t "_blank)

Use code [with caution.](/faq#coding)

1. **Install the required packages:**
   * Run the following commands:

Bash

pip install django

pip install django-mssql-backend

pip install pyodbc # If you'll be using it directly

# pip install openpyxl # or xlsxwriter (if you need Excel export)

Use code [with caution.](/faq#coding)

1. **Generate requirements.txt:**
   * Run the following command to create a requirements.txt file that lists all the installed packages and their versions:

Bash

pip freeze > requirements.txt

Use code [with caution.](/faq#coding)

Now you have a clean virtual environment with all the necessary packages installed. You can proceed with configuring your Django settings, defining your model, and running the migrations again.

**Remember:**

* Always activate your virtual environment before working on your Django project or running any Django commands.
* If you add any new packages to your project later, make sure to update your requirements.txt file using pip freeze > requirements.txt.

Feel free to ask if you have any further questions or need assistance with any of these steps. I'm here to help you throughout the process!

Sources and related content

[github.com](https://github.com/MATH-517/math_517_website" \t "_blank)

[github.com](https://github.com/MATH-517/math_517_website" \t "_blank)



gemini, i think i need Microsoft C++ Build Tools installed on my laptop similar to the server , so i ll leave it downloading, can we save this chat for later when i continue my django project? can there be a url or will i find it in your history?

Show drafts

Absolutely, you can save this chat for later reference!

Unfortunately, there's no direct URL to access specific chats in the current version. However, you have two options to retrieve it later:

1. **Copy and Paste:**

* You can select the entire chat content and copy it to a text editor or document on your computer.
* This way, you'll have a local copy of the conversation for future reference.

1. **Search Your Chat History:**

* When you return to the chat interface, you can use the search bar at the top to search for keywords or phrases from our conversation.
* This should help you locate the relevant parts of the chat.

**Remember:**

* The chat history might be cleared periodically, so it's best to save a local copy if you need to refer back to it after a longer period.

Once you have the Microsoft C++ Build Tools installed and are ready to continue with your Django project, feel free to start a new chat and I'll be here to assist you!

**Upload an image**

This prompt requires you to add an image. To add an image, select .

Got it

**Need a little help with this prompt?**

Power up your prompt and Gemini will expand it to get you better results

Got it

Gemini may display inaccurate info, including about people, so double-check its responses. [Your privacy & Gemini Apps Opens in a new window](https://support.google.com/gemini?p=privacy_notice)



Google apps

Google Account

Eng. Omar Essam

eng.omar.e@gmail.com

Use microphone

Expand text

More

Collapse

Learn more

Add files

Open in Lens

Collapse menu

Help

Gemini Apps Activity

Settings

Edit text