Project

Enterprise Resource Planner (E.R.P.)

Team Members

Mausam Kumar Shah(077bei023)

Rohan Bade (077bei036)

Abstract:

Managing Financial Transactions and User Profiles in a Web Application

In today's digital age, efficient management of financial transactions and user profiles is vital for individuals and businesses alike. This project aims to create a comprehensive web application that enables users to seamlessly record, categorize, and analyze their financial transactions while also providing a platform to manage their user profiles.

The project's core functionality revolves around two primary entities: **Transactions** and **User Profiles**. The **Transaction** entity captures the details of financial activities, including transaction type (income or expense), date and time, subject, cost, and optional remarks. This comprehensive dataset empowers users to maintain a clear record of their financial interactions and monitor their financial health over time.

User profiles, represented by the **Profile** entity, enable users to personalize their accounts with profile images. The one-to-one relationship between users and profiles ensures a secure and seamless integration of user data.

The application harnesses the power of the Django web framework, employing its Object-Relational Mapping (ORM) capabilities to manage database interactions effortlessly. Through a user-friendly interface, users can add, view, edit, and delete transactions, categorize them as income or expense, and provide contextual information using remarks. The datetime attribute assists in sorting and arranging transactions for better analysis.

Additionally, the project underscores the significance of code modularity and reusability by adopting Django's app architecture. This architecture promotes clean code separation and encapsulates specific functionalities, ensuring maintainability and scalability.

Introduction:

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It includes various components and features that make it easy to build web applications efficiently. One of the crucial aspects of Django is its database handling capabilities. Let's explore Django's database-related features in depth.

1. **Object-Relational Mapping (ORM):**

   Django provides an Object-Relational Mapping (ORM) system that abstracts the interaction with databases. Instead of writing raw SQL queries, you work with Python objects that are automatically mapped to database tables. This abstraction simplifies database interactions and makes the code more readable and maintainable.

2. **Database Configuration:**

   In the Django project's settings, you can configure the database connection details such as database engine (SQLite, PostgreSQL, MySQL, etc.), host, port, username, and password. Django supports multiple databases simultaneously.

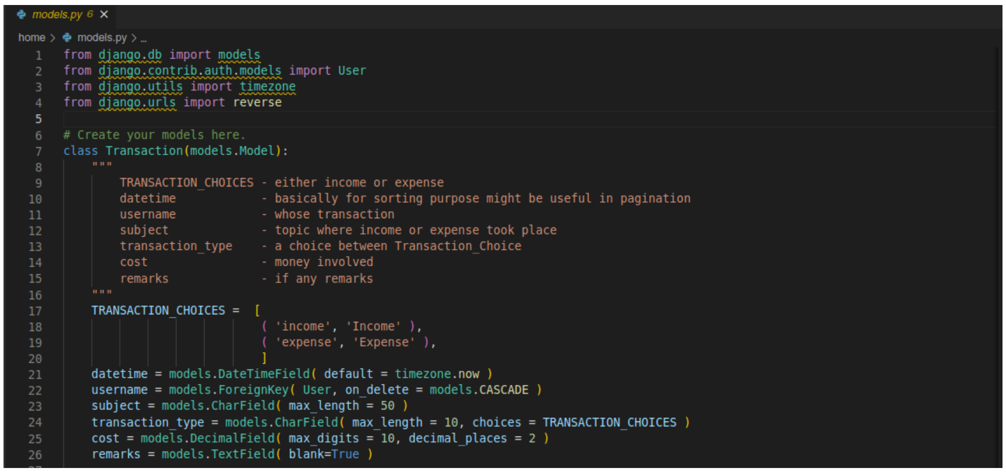
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

3. **Models:**

   In Django, a model is a Python class that defines the structure of a database table. Each attribute of the model class represents a field in the corresponding table. The Django ORM automatically generates SQL queries to create, modify, and query the database tables based on the model definitions.

```
from django.db import models
```

```
class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField()

    def __str__(self):
        return self.name
```



models.py 6 ×

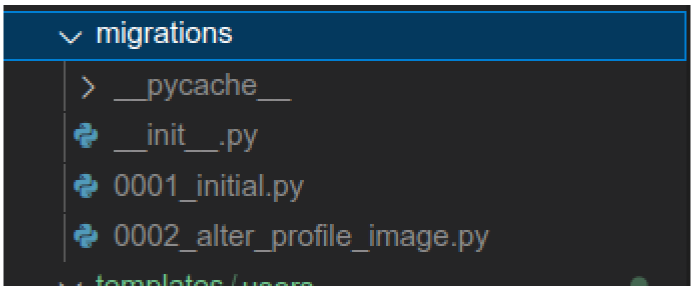
home > models.py > ...

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
from django.urls import reverse

# Create your models here.
class Transaction(models.Model):
    """
        TRANSACTION_CHOICES - either income or expense
        datetime            - basically for sorting purpose might be useful in pagination
        username            - whose transaction
        subject             - topic where income or expense took place
        transaction_type    - a choice between Transaction_Choice
        cost                - money involved
        remarks             - if any remarks
    """
    TRANSACTION_CHOICES =  [
                            ( 'income', 'Income' ),
                            ( 'expense', 'Expense' ),
                            ]
    datetime = models.DateTimeField( default = timezone.now )
    username = models.ForeignKey( User, on_delete = models.CASCADE )
    subject = models.CharField( max_length = 50 )
    transaction_type = models.CharField( max_length = 10, choices = TRANSACTION_CHOICES )
    cost = models.DecimalField( max_digits = 10, decimal_places = 2 )
    remarks = models.TextField( blank=True )
```

4. Migrations:

Django's migration system allows you to evolve your database schema over time without manually writing SQL. Migrations are Python scripts that describe changes to the database schema, such as adding or modifying tables and columns. You create migrations based on changes to your models and then apply those migrations to the database.

```
# Create a migration
python manage.py makemigrations


# Apply migrations
python manage.py migrate
```



5. Querysets:

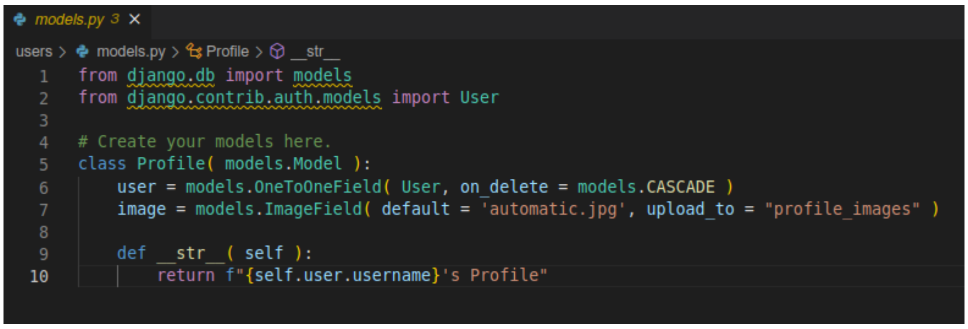
Django provides a QuerySet API to perform database queries. QuerySets allow you to retrieve, filter, sort, and manipulate data from the database using a chainable API similar to SQL. They provide a convenient way to interact with the database without writing raw SQL queries.

```
# Retrieve all products
products = Product.objects.all()


# Filter products based on a condition
cheap_products = 
Product.objects.filter(price__lt=50)
```

6. Relationships:

Django supports various types of database relationships, including OneToOneField, ForeignKey, and ManyToManyField. These relationships are used to establish connections between different model classes.

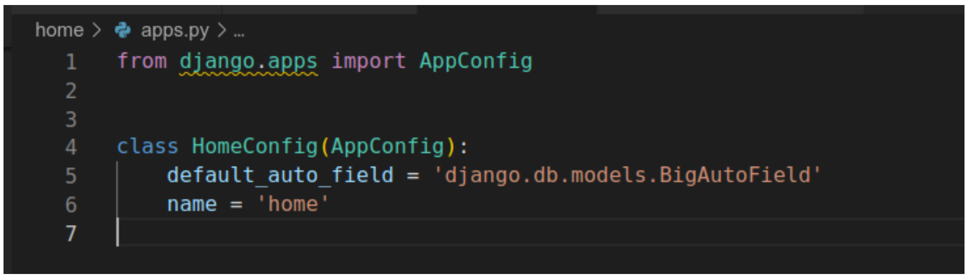
```
class Order(models.Model):
    customer = models.ForeignKey(Customer, 
on_delete=models.CASCADE)
    products = models.ManyToManyField(Product)
```



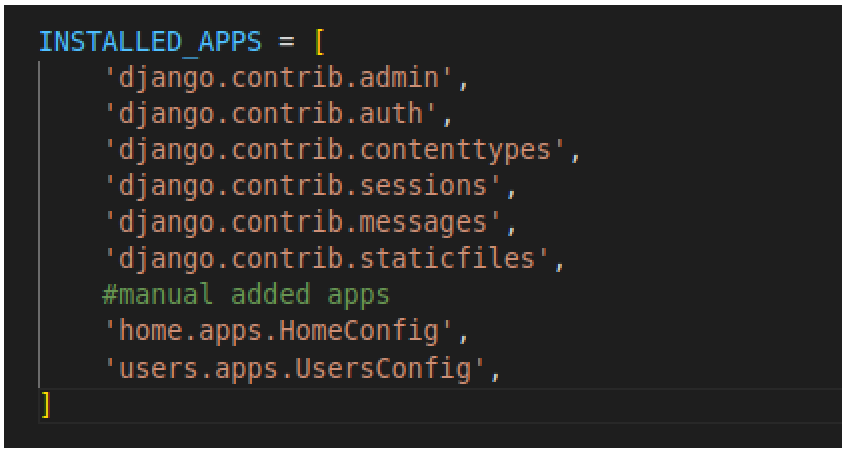
```python
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class Profile( models.Model ):
    user = models.OneToOneField( User, on_delete = models.CASCADE )
    image = models.ImageField( default = 'automatic.jpg', upload_to = "profile_images" )

    def __str__( self ):
        return f"{self.user.username}'s Profile"
```

7. **Apps in Django:**

In Django, an "app" refers to a self-contained module that encapsulates a specific functionality or set of related features within a larger web application. Django apps are designed to promote modularity, reusability, and maintainability by organizing code into separate components. Each app can be developed independently and then integrated into a Django project to build a complete web application.



```python
from django.apps import AppConfig


class HomeConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'home'
```



```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    #manual added apps
    'home.apps.HomeConfig',
    'users.apps.UsersConfig',
]
```

8. **Raw SQL Queries:**

While Django's ORM is powerful, there might be cases where you need to execute raw SQL queries. Django provides facilities to execute custom SQL queries while still maintaining the database connection and transaction management.

```
    from django.db import connection


    def custom_query():
        with connection.cursor() as cursor:
            cursor.execute("SELECT * FROM
myapp_mymodel")
            results = cursor.fetchall()
        return results
```

Project Goals and Objectives:

Our Project goal was to make use of the database concepts and apply them to web development. We put our least effort on UI whereas the database and the backend were the key areas of interest. Django made a lot of sense after we learnt SQL and making models in Django eased our way. So following were our goals:

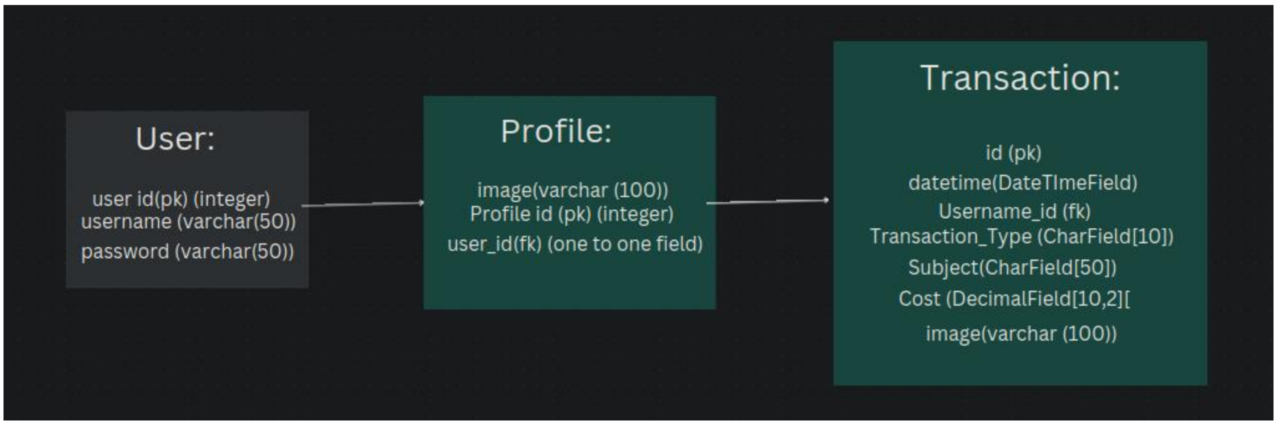
1. To get simple yet elegant UI using HTML, CSS, JS(to some extent).
2. Imply concepts such as primary key, foreign key, query, relationships on to database areas.
3. Learn and apply CRUD (Create, Read ,Update, Delete) in our application.
4. Use git for version tracking.
5. Be familiar with json files.
6. Use Login/ session handling, Registration and proper rules setting.
7. Use pagination to handle pages of 4 in each (pagination=5)
8. Use Viewing of list, detail etc

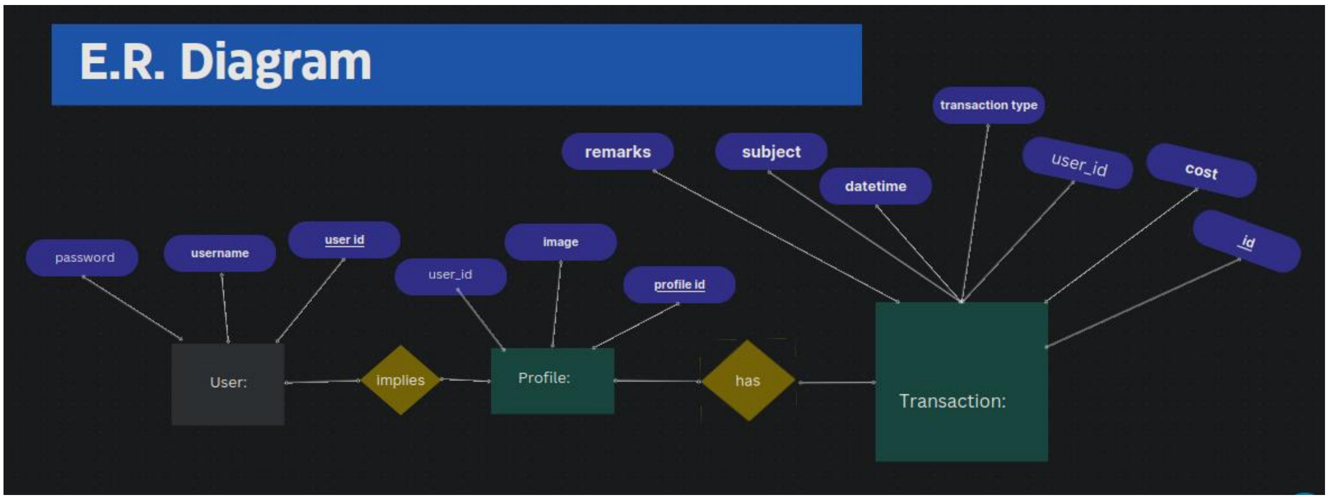
Technologies Used:

We used following technologies:

- **Git**
- **HTML, CSS, JS**
- **Django**

Architecture:





This section is concerned with E.R. Diagram.

- **Entities:**
- **User Entity:** Represents users of the application. Each user has a unique `id` (primary key) and a `username`.
- **Transaction Entity:** Represents individual transactions. Each transaction has a unique `id` (primary key), a `datetime` indicating when the transaction occurred, a `subject` describing the transaction's topic, a `transaction_type` representing whether it's an income or expense, a `cost` indicating the money involved, `remarks` for any additional comments, and a foreign key `username_id` that relates the transaction to a specific user.
- **Profile Entity:** Represents user profiles, with each user having one associated profile. Profiles have a unique `id` (primary key) and an `image` field that stores an uploaded image. The `user_id` field indicates the associated user through a one-to-one relationship.
- **Relationships:**

- **One-to-One Relationship (User to Profile):** Each user can have only one profile, and each profile is associated with one user. This relationship is represented by the `user_id` field in the `Profile` entity.
- **One-to-Many Relationship (User to Transaction):** Each user can have multiple transactions, but each transaction is associated with only one user. This relationship is represented by the `username_id` field in the `Transaction` entity.

Pending Issues:

- UI optimization and reponsiveness
- If we get good outliers free, well processed data, we might implement an AI model to replicate the learning.
- Crispy forms- Django utilities can be implemented
- Ability to work with excel sheets
- Ability to take user query in certain manner.

Conclusion:

Thus we have learnt a lot from this project. We were able to bring out our time and effort. Learn skills and apply them further. We thank Lecturer Bibha Sthapit Ma’am for empowering us with this opportunity.