Django Project Documentation

Django MVT Architecture

The Django MVT (Model-View-Template) architecture follows a clear separation of concerns, with models handling data persistence, views handling business logic, and templates handling presentation.

The process flow is as follows:

```
Browser Request -> URLs (urls.py) -> Views (views.py) -> Templates (HTML)
```

Getting Started

1. Open the project folder in your preferred code editor or IDE.

2. Apply database migrations:

   ```
   python manage.py makemigrations
   ```

   The `makemigrations` command is used to create new migration files based on the changes you have made to your models. It generates migration files that describe the changes to be applied to the database schema.

   ```
   python manage.py migrate
   ```

   The `migrate` command applies the migrations to the database, creating or modifying tables and columns as per the changes in the models.

3. Start the development server:

   ```
   python manage.py runserver
   ```

   This command starts the Django development server, allowing you to access and test the application locally.

Settings

Database Configuration

PostgreSQL

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'your_database_name',
        'USER': 'your_username',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

MySQL

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'your_database_name',
        'USER': 'your_username',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Azure File Storage Configuration

```
AZURE_ACCOUNT_NAME = env('AZURE_ACCOUNT_NAME')
AZURE_ACCOUNT_KEY = env('AZURE_ACCOUNT_KEY')
AZURE_CUSTOM_DOMAIN = f'{AZURE_ACCOUNT_NAME}.blob.core.windows.net'
AZURE_LOCATION = 'datagridztask'
```

```
AZURE_CONTAINER = 'datagridztask'

DEFAULT_FILE_STORAGE = 'dagz_project.custom_azure.AzureMediaStorage'
```

HTML Pages

- `dashboard.html` : Displays the user dashboard, showing summary statistics and visualizations of invoice data.
- `login.html` : Renders the login page for user authentication.
- `register.html` : Renders the user registration page.
- `pdfview.html` : Allows users to view PDF invoices and navigate through them, with options to filter invoices by status and date range.
- `invoice_data.html` : Displays detailed invoice data with filtering options for date range, customer name, and vendor. It presents the total value of invoices and lists customers and vendors based on the filtered data.
- `user_profile.html` : Allows users to view and update their profile information, including name, email, and password. It also provides fields for additional details like host, email, and password.

Models

CustomUser

The `CustomUser` model is a custom user model that extends `AbstractBaseUser` and `PermissionsMixin`. It has the following fields:

- `email` (EmailField): The email address of the user, which serves as the unique identifier.
- `name` (CharField): The name of the user.
- `host` (CharField, nullable): The host name or IP address.
- `eemail` (EmailField, nullable): An additional email field.
- `epassword` (CharField, nullable): A password field.
- `is_active` (BooleanField): Indicates whether the user account is active.
- `is_staff` (BooleanField): Indicates whether the user has staff privileges.

KS61

The `KS61` model is used to store invoice data extracted from PDFs. It has the following fields:

- `Registered_Business_Address` (JSONField, nullable): The registered business address.
- `Telephone` (CharField, nullable): The telephone number.
- `Email` (CharField, nullable): The email address.

- `Company_Number` (CharField, nullable): The company number.
- `Vat_Number` (CharField, nullable): The VAT number.
- `Trading_Address` (JSONField, nullable): The trading address.
- `Business_Name` (CharField, nullable): The business name.
- `Account_Terms` (CharField, nullable): The account terms.
- `Invoice_Date` (CharField, nullable): The invoice date.
- `Invoice_Number` (CharField, nullable): The invoice number.
- `Date` (CharField, nullable): The date.
- `created_at` (DateField): The date and time when the record was created.
- `Authorisation_Number` (CharField, nullable): The authorization number.
- `Make` (CharField, nullable): The make or manufacturer.
- `Model` (CharField, nullable): The model.
- `Registration` (JSONField, nullable): The registration details.
- `Description_of_Work` (JSONField, nullable): The description of the work performed.
- `Price` (JSONField, nullable): The price or cost.
- `Net_Total` (DecimalField, nullable): The net total amount.
- `Vat_at_20_Percent` (DecimalField, nullable): The VAT amount at 20 percent.
- `Invoice_Total` (DecimalField, nullable): The total invoice amount.
- `Total_Payable` (CharField, nullable): The total payable amount.
- `Payment_Due` (CharField, nullable): The payment due date or information.
- `Customer_Name` (CharField, nullable): The customer name.
- `file` (OneToOneField, nullable): The associated `Invoices` instance.

Invoices

The `Invoices` model is used to store uploaded invoice files and their status. It has the following fields:

- `invoice` (FileField): The uploaded invoice file.
- `date` (DateField): The date when the invoice was uploaded.
- `status` (CharField): The status of the invoice (e.g., ‘Processed’, ‘Rejected’).

Product

The `Product` model is used to store product descriptions and prices. It has the following fields:

- `description` (CharField): The product description.
- `price` (DecimalField): The product price.

URLs and View Functions

- ‘’ - `dashboard`
    - Path: ‘’
    - View Function: dashboard
    - Description: Renders the dashboard view.
- ‘login’ - `login_view`
    - Path: ‘login’
    - View Function: login_view
    - Description: Handles the user login functionality.
- ‘register’ - `register_user`
    - Path: ‘register’
    - View Function: register_user
    - Description: Handles the user registration functionality.
- ‘logout’ - `LogoutView.as_view(next_page="login")`
    - Path: ‘logout’
    - View Function: LogoutView.as_view(next_page=“login”)
    - Description: Handles the user logout functionality and redirects to the login page.
- ‘dashboard’ - `dashboard`
    - Path: ‘dashboard’
    - View Function: dashboard
    - Description: Renders the dashboard view.
- ‘pdf_view/’ - `pdf_view`
    - Path: ‘pdf_view/’
    - View Function: pdf_view
    - Description: Renders the PDF view without an invoice ID.
- ‘pdf_view/<int:invoice_id>/’ - `pdf_view`
    - Path: ‘pdf_view/<int:invoice_id>/’
    - View Function: pdf_view

    - Description: Renders the PDF view with a specific invoice ID.

- ‘file_upload’ - `file_upload`

    - Path: ‘file_upload’
    - View Function: file_upload
    - Description: Handles the file upload functionality.

- ‘invoice_data’ - `invoice_data`

    - Path: ‘invoice_data’
    - View Function: invoice_data
    - Description: Handles the invoice data functionality.

- ‘user_profile’ - `user_profile`

    - Path: ‘user_profile’
    - View Function: user_profile
    - Description: Renders the user profile view.

- ‘rej_to_proc/<int:id>/’ - `direct_data_extractor`

    - Path: ‘rej_to_proc/<int:id>/’
    - View Function: direct_data_extractor
    - Description: Handles the direct data extraction functionality for a specific invoice ID.

- ‘upload_csv’ - `upload_csv` (from utils.py)

    - Path: ‘upload_csv’
    - View Function: upload_csv (from utils.py)
    - Description: Handles the CSV upload functionality.

- ‘export_csv’ - `export_csv` (from utils.py)

    - Path: ‘export_csv’
    - View Function: export_csv (from utils.py)
    - Description: Handles the CSV export functionality.

Additional Functions

- `data_extractor`

    - Description: Extracts data from a PDF file using regular expressions and the tabula library. It retrieves various fields such as telephone, email, company number, VAT number, business name, account terms, invoice date, invoice number, net total, VAT, invoice total, total payable, and payment due. The extracted data is then stored in the `KS61` model.
- `validate`
    - Description: Validates the extracted invoice data by checking if the product descriptions and prices match the records in the `Product` model. It also verifies if the sum of prices matches the net total and if the VAT calculation is correct.
- `upload_csv`
    - Description: Handles the CSV upload functionality. It reads the CSV file, maps the headers to the corresponding fields in the `KS61` model, and bulk creates `KS61` objects based on the CSV data.
- `download_csv`
    - Description: Generates a CSV file containing data from the specified queryset (e.g., `KS61` objects) and returns an HTTP response with the CSV file as an attachment.
- `export_csv`
    - Description: Invokes the `download_csv` function with the `KS61` queryset to generate and download a CSV file containing all invoice data.