**Cheque Pay – Cheque Printing Solution**

**Version 1.1.0-fr001** 

**Table of Contents**

Table of Contents

[Overview 3](#_Toc199509480)

[System Requirements 5](#_Toc199509481)

[Django Settings 8](#_Toc199509482)

[Module Description 9](#_Toc199509483)

[Login 9](#_Toc199509484)

[Companies 10](#_Toc199509485)

[Banks 11](#_Toc199509486)

[Payees 12](#_Toc199509487)

[Currencies 13](#_Toc199509488)

[Cheque Templates 14](#_Toc199509489)

[Issued Cheques 15](#_Toc199509490)

[Reports 16](#_Toc199509491)

## Overview

**Cheque Pay - Cheque Printing Solution**

The **Cheque Pay** application is a comprehensive cheque printing solution tailored for businesses, accountants, and individuals who regularly issue bank cheques. With increasing digitalization, manual cheque writing can be time-consuming and prone to errors. Cheque Pay resolves these challenges by offering a user-friendly, precise, and customizable platform that ensures professional cheque printing with minimal effort.

Designed to support multiple bank formats and institutions, this solution empowers users to print cheques that conform to bank-specific layouts. Whether the user is printing a single cheque or handling bulk cheque issuance, Cheque Pay streamlines the process with precision and flexibility.

**Key Features:**

* **Customizable Templates**: Upload scanned images of your bank's cheque leaves and define the positions for fields like Date, Payee, Amount in Numbers, Amount in Words, and Signature using a drag-and-drop interface.
* **Multi-Company Management**: Handle cheque issuance for multiple companies under a single user account. Each company can maintain its own templates, bank accounts, and payee information.
* **Multi-Currency Support**: Supports printing in different global currencies (e.g., INR, USD, EUR), including localized amount-in-words generation.
* **Preview & Print**: Users can preview cheques with actual data before printing, reducing errors and avoiding cheque wastage.
* **Cloud or On-Premise Deployment**: Flexibility to deploy Cheque Pay on a local server or cloud, based on organizational requirements.
* **Security & Licensing**: Licensing features are built-in to control usage, ensuring that only authorized users and devices can access company data.

**Customization:**

Cheque Pay allows advanced customization of fonts, sizes, styles (bold, italic), and positioning for each text field. These settings can be saved per template, offering a high degree of flexibility for institutions with unique formatting standards.

Additionally, users can add background logos, watermark overlays, or include authorized signatory lines where applicable. The application ensures that each cheque appears uniform, professional, and aligned to the bank's clearance standards.

**Use Cases:**

* **Small and Medium Enterprises (SMEs)**: Manage multiple accounts, banks, and payees without depending on pre-printed cheques.
* **Chartered Accountants & Finance Professionals**: Print cheques for multiple clients while maintaining a record of issued cheques.
* **Personal Finance Management**: Individuals can track and print cheques with ease, avoiding handwriting errors and maintaining clarity.

**Technical Notes:**

* Built using Django (Python) on the backend for high reliability and security.
* JavaScript-driven front-end for dynamic UI interactions, including drag-and-drop text placement.
* MS SQL Server or SQLite for flexible database options depending on deployment scale.
* Compatible with standard A4/cheque printers and supports margin calibration for precise output.

**Advantages of Using Cheque Pay:**

* Reduces cheque handling time by automating formatting and positioning.
* Minimizes human errors such as spelling mistakes, misplaced text, or wrong amounts.
* Centralizes cheque records for better auditing, traceability, and reporting.
* Ensures professional output, enhancing brand credibility and presentation.
* Easy-to-use interface requiring no technical background or design skills.

## System Requirements

* Operating System: Windows, MacOS, or Linux
* Tools: Atom or Visual Studio Code or Visual Studio , MS SQL Server

Management Studio 18

* Web Server: Windows server
* Database: Sqlite3
* Web Browsers : Google Chrome, Mozilla Firefox
* Dependencies:

Python==3.10**, Django==5.1.2, djangorestframework==3.15.2, django-formtools==2.5.1, python-dotenv==1.0.1, Cython==3.0.12, pyinstaller-hooks-contrib==2025.1, whitenoise==6.8.2, waitress==3.0.2, pandas==2.2.3, openpyxl==3.1.5, pdfkit==1.0.0, opencv-python==4.11.0.86, rembg==2.0.64, PyMatting==1.1.13, numpy==1.26.4, scipy==1.15.2, scikit-image==0.25.2, onnxruntime==1.15.0, numba==0.61.0, llvmlite==0.44.0, requests==2.32.3, urllib3==2.3.0, idna==3.10, certifi==2024.12.14, charset-normalizer==3.4.1.**Architecture

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Its architecture follows the Model-View-Controller (MVC) pattern, with some variations. The primary components of Django architecture include:

1. **Model:**
   * The Model is responsible for managing the data and the business logic of the application.
   * It defines the structure of the database, including tables, fields, and relationships.
   * Django's Object-Relational Mapping (ORM) system allows developers to interact with the database using Python code rather than SQL.
2. **View:**
   * The View is responsible for processing user requests and returning appropriate responses.
   * In Django, views are Python functions or classes that take a web request as input and return a web response.
   * Views interact with the Model to retrieve or manipulate data and use templates to generate HTML for the response.
3. **Template:**
   * The Template is responsible for generating dynamic HTML content based on the data provided by the View.
   * Templates use Django's template language to embed Python-like code for dynamic content rendering.
   * Separating the template layer from the View allows for better code organization and maintenance.
4. **Controller (URLconf):**
   * In Django, the Controller is represented by the URLconf (URL configuration).
   * URLconf maps URLs to specific views, determining which view should handle a particular request.
   * The **urls.py** file in a Django project defines the URL patterns and their corresponding views.
5. **Middleware:**
   * Middleware components are responsible for processing requests and responses globally before reaching the view or after leaving the view.
   * Examples of middleware include authentication middleware, session middleware, and error-handling middleware.
   * Middleware allows developers to add functionality to the request/response processing pipeline.
6. **Forms:**
   * Django provides a form-handling system that simplifies the process of collecting and validating user input.
   * Forms help in managing the HTML forms on the client side and processing the form data on the server side.
7. **Admin Interface:**
   * Django includes a built-in admin interface that allows developers to manage data in the application without directly interacting with the database.
   * The admin interface is customizable and automatically generated based on the models defined in the application.

Django follows the "Don't Repeat Yourself" (DRY) and "Convention Over Configuration" principles, aiming to reduce redundancy and make development more straightforward by providing sensible default configurations. The framework's architecture promotes a clean separation of concerns, making it easier to maintain and scale web applications.

## Django Settings

Django settings are the project’s main settings file which states all the important parameters for the project. Such as database connections, allowed hosts, static files, middleware and many more. Some of which are mentioned below,

* ALLOWED\_HOSTS = ['\*'] – Allows the project to be run on mentioned IPs.
* DEBUG = True – Allows the default debug process to be active
* INSTALLED\_APPS [ ‘chq\_pay’] – mentions installed app in the project.
* MIDDLEWARE = [ 'ChequePay.middlewares.AttachRequestToDBRouterMiddleware'] – Added for multiple database data access.
* DATABASES – This setting allows and defines the database connection for the project. Database settings are kept in a file called db\_config.py so that the settings can be updated for multiple connections.
  + from ChequePay.db\_config import DATABASES

DATABASES = DATABASES

* STATIC – Used for storing images and CSS files
  + STATIC\_URL = '/static/'
  + STATICFILES\_DIRS = [STATIC\_DIRS,] #Added by User
  + # STATIC\_ROOT = STATIC\_DIRS #By user for production
* MEDIA – To store the media input form the users.
  + MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')
  + MEDIA\_URL = "/media/"
* LOGIN URL – To mention the default login URL to jump back to when logout of the project.
  + LOGIN\_URL = ' /Chequepay/login/' #for the user login with license

## Module Description

There are different parts, comprising of which the project is made of. I will be describing them as the project flow. The name of the project is **ChequePay** and app name is **chq\_pay.** Every module consists -

* **Model** – These can be described as tables that are used in the database. Can be modified to be used into the project.
* **Views** – These are backend functions which is used to control and manipulate the functionality of any module.
* **URL** – These are the name s by which a particular page is been called and also what functions is acting on the particular page.
* **Templates** - These are the HTML files which are used to showcase the frontend of the project.
* **Forms** - Some of the views has also used **forms.py**  file which consists of different forms which can be used to make a form directly into templates without using certain HTML tags.

### Login

It is the basic page which allows the user to access the application by taking correct credentials. **Model**

* from django.contrib.auth.models import AbstractUser - this is default library which is used to make different types of users.
* ChqpayUser

**View (authentic\_views)**

* user\_login – to log in to the application
* user\_logout – to log out of the application

**URL**

* path('check\_reg\_update\_sett/', imp\_libs.check\_reg\_update\_sett, name = "check\_reg\_update\_sett")
* path('login/', authentic\_views.user\_login, name = "login"),
* path("logout/", authentic\_views.user\_logout, name="logout"),

**Templates**

* login.html

### Companies

**Model**

* Company\_Setup

**View (company\_views)**

* add\_company – to add new company
* company\_list – to showcase list of companies
* delete\_company – to delete the existing companies
* edit\_company – to edit the details of the company
* current\_company – to keep selected the current company

**URL**

* path('add\_company/', company\_views.add\_company, name='add\_company')
* path('company\_list/', company\_views.company\_list, name = "company\_list")
* path('edit\_company/', company\_views.edit\_company, name='edit\_company')
* path('delete\_company/<int:company\_id>/', company\_views.delete\_company, name='delete\_company')
* path('current\_company', company\_views.current\_company, name='current\_company')

**Templates**

* company\_list.html

### Banks

**Model**

* Banks

**View (bank\_views)**

* add\_bank – to add new bank
* bank\_list – to showcase list of banks
* delete\_bank – to delete the existing banks
* edit\_bank – to edit the details of the bank
* validate\_excel\_bank – to validate uploaded excel data
* save\_valid\_data\_bank – to save the validated data
* download\_sample\_bank – to download the sample excel to upload the data

**URL**

* path('add-bank/', bank\_views.add\_bank, name='add\_bank')
* path('bank\_list/', bank\_views.bank\_list, name = "bank\_list")
* path('edit-bank/', bank\_views.edit\_bank, name='edit\_bank')
* path('delete\_bank/<int:bank\_id>/', bank\_views.delete\_bank, name='delete\_bank')
* path('excel\_upload\_bank/validate/', bank\_views.validate\_excel\_bank, name='validate\_excel\_bank')
* path('excel\_upload\_bank/save/', bank\_views.save\_valid\_data\_bank, name='save\_valid\_data\_bank')
* path('download\_sample\_bank/', bank\_views.download\_sample\_bank, name='download\_sample\_bank')

**Templates**

* bank\_list\_new.html

### Payees

**Model**

* Payee

**View (payee\_views)**

* add\_payee – to add new payee
* payee\_list – to showcase list of payees
* delete\_payee – to delete the existing payees
* edit\_payee – to edit the details of the payee
* validate\_excel\_payee – to validate data from the uploaded excel to import
* save\_valid\_data\_payee – to save the validated data
* download\_sample\_payee – to download the sample excel to upload the data

**URL**

* path('add\_payee/', payee\_views.add\_payee, name='add\_payee')
* path('payee\_list/', payee\_views.payee\_list, name = "payee\_list")
* path('edit\_payee/', payee\_views.edit\_payee, name='edit\_payee')
* path('delete\_payee/<int:payee\_id>/', payee\_views.delete\_payee, name='delete\_payee')
* path('excel\_upload\_payee/validate/', payee\_views.validate\_excel\_payee, name='validate\_excel\_payee')
* path('excel\_upload\_payee/save/', payee\_views.save\_valid\_data\_payee, name='save\_valid\_data\_payee')
* path('download\_sample\_payee/', payee\_views.download\_sample\_payee, name='download\_sample\_payee')

**Templates**

* payee\_list\_new.html

### Currencies

**Model**

* Currencies

**View (currency\_views)**

* add\_currency – to add new currency
* currency\_list – to showcase list of currencies
* delete\_currency – to delete the existing currencies
* edit\_currency – to edit the details of the company

**URL**

* path('add\_currency/', currency\_views.add\_currency, name='add\_currency')
* path('currency\_list/', currency\_views.currency\_list, name = "currency\_list")
* path('edit\_currency/', currency\_views.edit\_currency, name='edit\_currency')
* path('delete\_currency/<int:currency\_id>/', currency\_views.delete\_currency, name='delete\_currency')

**Templates**

* currency\_list\_new.html

### Cheque Templates

**Model**

* ChequeTemplate
* ChequeText

**View (cheque\_views)**

* upload\_template – to add new template
* template\_list – to showcase list of templates
* delete\_template – to delete the existing templates
* edit\_template – to edit the details of the templates
* template\_detail – to showcase the template and text position details
* add\_text\_to\_template – to set the text positions of the certain text on the template
* reset\_text – to reset the text positions of the texts from the template
* delete\_text – to delete the text position set for the particular text

**URL**

* path('upload\_template/', cheque\_views.upload\_template, name = "upload\_template"),
* path('template\_list/', cheque\_views.template\_list, name = "template\_list"),
* path('edit-template/', cheque\_views.edit\_template, name='edit\_template'),
* path('template/<int:template\_id>/', cheque\_views.template\_detail, name='template\_detail'),
* path('delete\_temp/<int:template\_id>/', cheque\_views.delete\_template, name='delete\_template'),
* path('template/<int:template\_id>/add-text/', cheque\_views.add\_text\_to\_template, name='add\_text\_to\_template'),
* path('reset\_text/<int:temp\_id>/', cheque\_views.reset\_text, name='reset\_text'),
* path('delete\_text/<int:text\_id>/<int:temp\_id>/', cheque\_views.delete\_text, name='delete\_text'),

**Templates**

* currency\_list\_new.html

### Issued Cheques

**Model**

* ChequeIssue

**View (cheque\_issue)**

* cheque\_issue – to issue new cheque
* cheque\_issue\_list – to showcase list of issued cheques
* delete\_chequeissue – to delete issued cheques
* edit\_currency – to edit the details of the company
* get\_cheque\_text – to get the cheque’s text position
* approval – to approve/reject the issued chequed
* print\_cheque – to print the approved cheque

**URL**

* path('cheque\_issue/', cheque\_issue.cheque\_issue, name = "cheque\_issue")
* path('cheque\_issue\_list/', cheque\_issue.cheque\_issue\_list, name = "cheque\_issue\_list")
* path('edit\_cheque\_issue/', cheque\_issue.edit\_cheque\_issue, name='edit\_cheque\_issue')
* path('delete\_chequeissue/<int:chequeissue\_id>/', cheque\_issue.delete\_chequeissue, name='delete\_chequeissue')
* path('get\_cheque\_text', cheque\_issue.get\_cheque\_text, name='get\_cheque\_text')
* path('print-cheque/<int:cheque\_id>/', cheque\_issue.print\_cheque, name='print\_cheque')
* path('approval/<int:cheque\_id>/', cheque\_issue.approval, name='approval')

**Templates**

* cheque\_issue\_list.html
* print\_cheque.html

### Reports

**View (report\_issue)**

* reports – for the report generation
* export\_payee\_report – to download the CSV/XLS for the payee report
* export\_bank\_report – to download the CSV/XLS for the bank report

**URL**

* path('reports/', report\_views.reports, name='reports')
* path('reports/export-payee/<str:file\_format>/', report\_views.export\_payee\_report, name='export\_payee\_report')
* path('reports/export-bank/<str:file\_format>/', report\_views.export\_bank\_report, name='export\_bank\_report')

**Templates**

* reports.html
* bank\_report.html
* payee\_report.html