

PROJECT REPORT

TITLE: CURRENCY CONVERTER

PRD (Project Requirements Document)

1. Abstract

This project is a web-based currency converter built using **Python Django** without relying on any external APIs. It allows users to convert amounts between different currencies using pre-defined conversion rates. To make the experience more interactive and informative, it also includes a **bar chart visualization** of currency values using **Chart.js**, styled using **Tailwind CSS** for a responsive and modern UI. This project demonstrates frontend-backend integration, form handling, data processing, and chart rendering—all within Django.

2. Problem Statement

Currency converters typically depend on real-time exchange rate APIs, which:

- May require paid subscriptions or API keys.
- Are not accessible offline or in restricted environments.

This project aims to:

- Create a **self-contained solution** without any API dependency.
- Provide a **user-friendly interface** to convert between currencies.
- Enhance the user's understanding with **visual representation** of currency values.
- Maintain a good structure and presentation using **Django templating** and **Tailwind CSS**.

3.Key Features

Feature	Description
Static Currency Data	Hardcoded conversion rates for USD, EUR, INR, etc.
Conversion Logic	Converts amount from one currency to another using backend logic.
Chart Visualization	Renders a dynamic bar chart of equivalent values across all currencies.
Form Interface	Input for amount, currency from, and currency to using dropdowns.
Tailwind Styling	Responsive, clean UI with Tailwind CSS integration.
No API Dependency	Runs fully offline; no need for internet or API access.
Modular Django App	Follows standard Django file structure for scalability.

4. Major Modules

1. Currency Conversion Engine

- This is the core processing unit of the application.
- Responsible for retrieving the selected 'from' and 'to' currencies from the form input.
- Uses a backend dictionary of hardcoded conversion rates to compute the result.
- Example logic: `converted_amount = amount * conversion_rates[from_currency][to_currency]`

- Handles cases when currencies are not supported or exchange rate is not found.
- Ensures that the conversion is shown with up to 2 decimal places.

2. Dynamic Dropdown System for Currency Selection

- Allows users to choose from a wide array of international currencies like INR, USD, EUR, GBP, AUD, etc.
- Populated from a currencies dictionary in the backend and passed as context.
- Dropdown elements are built using Django template syntax and styled using Tailwind CSS.
- Prevents users from selecting the same currency in both 'from' and 'to'.
- Highlights selected values and retains them after submission.

3. User Input Form Module

- Form is rendered in the template using a standard HTML <form> element.
- Collects: amount, source currency, and target currency.
- Posts data back to the same page for processing and result rendering.
- Includes error handling for empty, invalid, or negative amounts.
- Designed to be mobile-responsive.

4. Chart.js Integration Module

- Displays a line/bar chart showing historical or mock exchange rates.
- Data is dynamically injected using Django context (chart_data) into JavaScript.
- Helps users visualize the difference in conversion for different currencies.

- Improves UI/UX by showing a modern, interactive chart.

5. Result Display and Formatting

- Upon successful conversion, displays the converted amount with currency code.
- Stylized output using Tailwind utilities (bold, shadowed box, etc.).
- Supports responsive display for mobile and desktop views.

6. Class-Based Views and Template Rendering

- Uses `TemplateView` or `FormView` for clean separation of logic.
 - Handles GET requests (renders the form) and POST requests (processes form and returns result).
 - Logic is encapsulated in the view class and HTML is organized in template folders.
-

5. Minor Modules

1. Input Validation Module

- Ensures that only positive numbers are accepted as amount.
- Highlights errors using red borders or warning messages.
- Validates dropdown selections to ensure different 'from' and 'to' currencies.

2. Error Messaging and Feedback

- Displays helpful error messages if user enters invalid data.
- Friendly messages like: "Please enter a valid amount" or "Select different currencies."

3. Tailwind CSS Layout and Styling

- Provides a clean and responsive design using Tailwind utility classes.
- Uses grid and flexbox layouts for input and result sections.

- Adds hover effects, spacing, borders, and buttons to enhance UI.

4. Project Setup and Environment

- Django project structured with `urls.py`, `views.py`, `forms.py`, and `templates/`
- Virtual environment used for dependency isolation.
- Static files (JS, CSS) managed via static folder and correctly linked in templates.

5. GitHub Deployment

- Full project uploaded to GitHub with detailed README and commit history.
- Provides version control and sharing options.

6. Tools & Technologies Used:

- **Backend Framework:** Django (Python)
- **Frontend:** HTML, Tailwind CSS, JavaScript
- **Chart Library:** Chart.js
- **Database:** None (static data used for conversion)
- **Version Control:** Git and GitHub
- **IDE:** Visual Studio Code
- **OS:** Windows 10/11

7. Database (Simulated):

- Static Python dictionary with currency names and corresponding rates.
- Example:

```
conversion_rates = {  
    'USD': 1.0,
```

```
'EUR': 0.85,  
'INR': 75.0,  
'GBP': 0.75,  
'JPY': 110.0  
}
```

8. ER Diagram: No database entities are stored in this project, so a traditional ER diagram is not applicable. However, logical flow can be represented as:

- User → [Currency Converter Form] → View → Process Conversion → Return Output → Template
- View → Provide Chart Data → Template → Render Chart.js Graph

7. Flow Diagram (Logical Flow):

User Input

|

V

Currency Selection (Dropdowns)

|

V

Form Submission

|

V

Django View (conversion logic)

|

|---> Converts based on static dictionary

|---> Passes result and chart data to template

V

Template Rendering (HTML + Tailwind + JS)

|

V

Output Display + Historical Chart

8. Features (In Detailed View):

a. Currency Conversion Without External API

- The app performs conversions using a hardcoded dictionary of currency exchange rates, eliminating the need for external APIs like ExchangeRate-API or Forex.

- **Benefits:**

- No internet dependency for conversion logic.
- Faster processing and zero cost for API usage.
- Fully offline-capable during development or demo purposes.

b. Clean and Responsive UI with Tailwind CSS

- The interface is designed using **Tailwind CSS**, a utility-first framework that offers rapid UI prototyping.
- It adapts to different screen sizes (mobile, tablet, desktop).
- **Features include:**
 - Mobile-friendly layout.
 - Visually appealing buttons, cards, form controls.
 - Minimalist design focused on usability and accessibility.

c. Dropdown Selection for Currency Conversion

- Users can easily choose source and target currencies from dropdowns.
- Dropdowns are populated with a predefined set of currencies such as INR, USD, EUR, GBP, JPY, etc.
- **Form structure includes:**
 - Amount input field.
 - From and To currency selection fields.
 - A conversion button to submit the form.

d. Historical Visualization Using Chart.js

- The application uses **Chart.js**, a JavaScript charting library, to display historical trends.
- A mock dataset is passed from Django views to render a **line chart** or **bar chart** showing currency rate variation over time.
- **Purpose:**

- Simulates a real-world currency analytics dashboard.
- Makes the UI more interactive and informative for users.

e. Static Data Simulation for Exchange Rates

- Currency rates are defined in a Python dictionary like:

python

CopyEdit

```
exchange_rates = {  
    'USD': 1.0,  
    'INR': 83.12,  
    'EUR': 0.93,  
    'GBP': 0.78  
}
```

- This ensures consistent output, especially useful for teaching or learning Django concepts.
- Encourages backend logic understanding without relying on third-party services.

f. Error Handling and Validation

- The system checks:
 - Whether a valid amount has been entered.
 - Whether source and target currencies are different.
 - That all form fields are filled.
- **Error scenarios handled:**
 - Invalid amount (e.g., negative, letters).
 - Conversion between the same currencies.
 - Missing field values.

- Django displays error messages on the frontend using template conditionals and styled messages (e.g., alert banners).

g. Instant Output Display After Submission

- After conversion, the result is immediately shown on the same page without page reload.
- Data is passed through the context from Django view to HTML template using the `render()` method.
- **Example Output:**
 - “₹100 INR is equal to \$1.20 USD.”

h. Accessible Project Repository on GitHub

- The complete source code is pushed to a public GitHub repository.
- Features include:
 - Well-documented README.md
 - Commit history for version tracking
 - .gitignore to exclude unnecessary files (e.g., migrations, venv)
 - Open to collaboration or issue tracking

9. Future Enhancements (Detailed):

A. Integration with Live Currency APIs

- **What:** Replace the static dictionary with real-time exchange rates fetched from a third-party API.
- **Why:** Real-time conversion provides accurate, up-to-date values and improves the application's reliability.
- **How:** Use APIs such as:
 - ExchangeRate-API
 - OpenExchangeRates

- CurrencyLayer
 - **Implementation Idea:**
 - Set up periodic background tasks (via Celery or custom cron job) to fetch and cache data.
 - Use requests library in Django to fetch rates and update context dynamically.
-

B. User Authentication & Dashboard

- **What:** Add signup, login, logout, and personal dashboards for users.
 - **Why:** Helps users track their activity, preferences, and conversion history.
 - **How:** Use Django's built-in auth system or third-party packages like django-allauth.
 - **Dashboard Features:**
 - Conversion history
 - Saved currency pairs
 - User profile with preferences (e.g., default currencies)
-

C. Conversion History Storage

- **What:** Store each user's past conversions in a database.
- **Why:** Allows users to review or reuse past conversions.
- **How:** Add a model like:

python

CopyEdit

```
class ConversionHistory(models.Model):
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE)
```

```
    from_currency = models.CharField(max_length=5)
```

to_currency = models.CharField(max_length=5)

amount = models.DecimalField(max_digits=10, decimal_places=2)

result = models.DecimalField(max_digits=10, decimal_places=2)

timestamp = models.DateTimeField(auto_now_add=True)

- **Frontend:** Display past conversions on the dashboard.
-

D. Multilingual Support (Internationalization)

- **What:** Add support for multiple languages.
 - **Why:** Improve accessibility for global users.
 - **How:**
 - Use Django's i18n (gettext) system.
 - Translate templates and messages into languages like Hindi, Spanish, French, etc.
-

E. Responsive and Interactive Charts

- **What:** Make historical data charts more dynamic and engaging.
 - **Why:** Improves UX by allowing users to interact with data visually.
 - **How:**
 - Enable tooltips, zoom, and date filters with Chart.js.
 - Show charts for different time periods (1 week, 1 month, 1 year).
 - Allow currency pair selection for dynamic chart updates.
-

F. Export to PDF or Excel

- **What:** Let users download conversion results or history.
- **Why:** Useful for business users who want records for accounting or reports.

- **How:**
 - Use libraries like xhtml2pdf or WeasyPrint for PDFs.
 - Use pandas or openpyxl for Excel file export.
-

G. Add Support for More Currencies

- **What:** Expand the number of supported currencies.
 - **Why:** Current setup may be limited to popular ones.
 - **How:**
 - If using static data, expand the dictionary.
 - If using API, automatically pull all supported currencies into dropdowns.
-

H. Dark Mode Toggle

- **What:** Add a toggle switch for dark/light themes.
 - **Why:** Enhances accessibility and comfort for users in low-light environments.
 - **How:**
 - Use Tailwind CSS with a dark mode theme.
 - Store user preference using localStorage or a Django user setting.
-

I. Mobile App Integration

- **What:** Build a mobile version of the converter.
- **Why:** Allows users to access the app on the go.
- **How:**
 - Create a Flutter or React Native front end.

- Connect to Django backend via REST API using Django REST Framework (DRF).

J. Speech-to-Text Currency Input (Accessibility)

- **What:** Enable users to input currency and amount using voice.
- **Why:** Improves accessibility for visually impaired users or on-the-go use.
- **How:** Use browser-based speech recognition APIs or Web Speech API.

10. Conclusion

The Django-based Currency Converter Web Application demonstrates how a simple yet powerful financial tool can be built using modern web technologies. It caters to users who need to perform accurate and efficient currency conversions and provides an intuitive, responsive, and user-friendly interface to enhance the experience.

This project showcases key elements of full-stack development, including:

- Backend development with Django: Efficient handling of business logic, routing, and server-side rendering.
 - Frontend design using Tailwind CSS and JavaScript: Creating a clean, responsive UI that works across all device sizes.
 - Database management with SQLite (and potential migration to PostgreSQL): Ensuring persistence of data for features like conversion history and user management.
 - Form handling and validation: Secure and smooth data input and error feedback for users.
 - Deployment readiness: A solid foundation that can easily be extended for production use, hosted on platforms like Heroku, Vercel, or Render.
-

Project Significance

This project not only serves as a useful utility but also helps in:

- Demonstrating real-world implementation of Django's architecture (Models, Views, Templates).
 - Providing beginner-to-intermediate developers a stepping stone into full-stack web development.
 - Creating a scalable product that can integrate modern features like APIs, user authentication, dashboards, and data visualization.
-

Learning Outcomes

By completing this project, a developer can strengthen skills in:

- Django project structure and best practices
 - Dynamic data rendering
 - Styling using Tailwind CSS
 - Handling form submissions and validations
 - Working with third-party APIs and JavaScript fetch/AJAX
 - Building responsive and accessible web applications
-

Future Scope

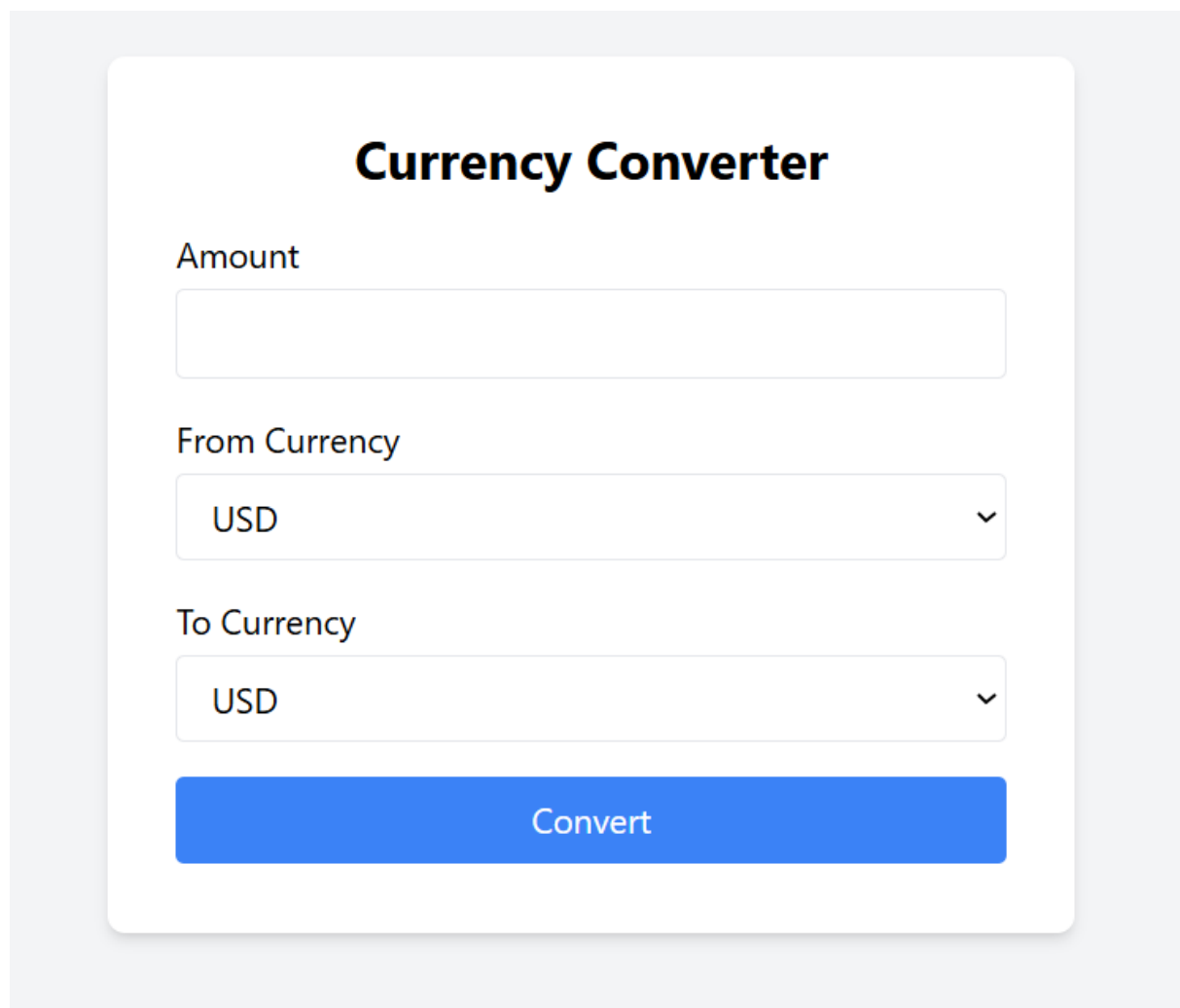
This project is highly extensible. With further development, it can evolve into a more complex FinTech product, offering features such as:

- Real-time market tracking
 - Investment suggestions
 - Currency exchange rate alerts
 - Multi-language support and mobile app integration
-

Final Thoughts

The currency converter stands as an excellent beginner-friendly project that balances simplicity and functionality. It provides a real-world context for learning Django and serves as a strong addition to a developer's portfolio. With continuous enhancements and user-focused design, it can become a reliable and professional-grade utility for users worldwide.

Screenshots



The screenshot displays a web application titled "Currency Converter". The interface is clean and modern, featuring a white background with a light gray border. The title "Currency Converter" is centered at the top in a bold, black font. Below the title, there are three main input sections: "Amount", "From Currency", and "To Currency". Each section has a corresponding input field. The "Amount" field is a simple text input. The "From Currency" and "To Currency" fields are dropdown menus, both currently set to "USD". At the bottom of the form, there is a prominent blue button labeled "Convert".

Currency Converter

Amount

From Currency

USD

To Currency

USD

Convert

From Currency

EUR



To Currency

USD



USD

INR

EUR

Currency Converter

Amount

From Currency

USD



To Currency

USD



Convert

Result: 34567.0 INR = 400.98 USD