



Computer and Systems Engineering [CSE]

Parallel and Distributed Systems [CSE352s]

Project: FlipFlow

Team Members

Name	ID
Mostafa Hamada Hassan Ahmed	2100587
Zeyad Magdy Roushdy	2100393
Mina Nasser Shehata	2100370
Ahmed Ashraf Ahmed	2100476
Mina Antony Samy Fahmy	2101022
Mark Wagdy Wadie	2100822
Mohamed Montasser Ibrahim	2100416
George Joseph Basilious Tawadrous	2100261
Youssef Hatem Ahmed	2101052

Contents

1) Introduction	5
1.1) Key features	5
2) Target Beneficiaries of the Project	7
3) Adopted Programming Languages and Technologies	7
4) System Architecture	9
4.1) Directory Structure (Simplified)	9
4.2) Architecture Explanation	10
5) Application Level Protocol	13
6) Distributed Database Design	16
6.1) Design Overview	16
6.2) Justification for the Distributed Design	16
6.3) Database schema	16
6.3.1) Item_category	16
6.3.2) Item_item	17
6.3.3) accounts_profile	17
6.3.4) Market_transaction	18
7) Time Plan	19
7.1) Planning phase	19
7.2) Development Phase	19
7.3) Testing Phase	20
7.4) Documentation Phase	20
8) Testing	21
8.1) Component Testing	21
8.2) System Testing	21
8.3) Testing Results	21
9) Needed Resources	22

10) End-User Guide	23
10.1) Running the Server	23
10.1.1) Install the requirements	23
10.1.2) Run commands open server.py	23
10.1.3) Run the Server	23
10.2) FlipFlow Walkthrough	23
11) Team Member Contributions	34
References	35

List of Tables

Table 1 Team Member Contributions	34
---	----

List of Listings

Listing 1 Directory Structure (Simplified)	9
--	---

List of Figures

Figure 1 System Architecture Diagram	11
Figure 2 Item Management via API	13
Figure 3 Creating Items with JSON	14
Figure 4 FlipFlow Timeline	19
Figure 5 Planning Phase Timeline	19
Figure 6 Development Phase Timeline	19
Figure 7 Testing Phase Timeline	20
Figure 8 Documentation Phase Timeline	20
Figure 9 Signup page	23
Figure 10 User Profile Page	24
Figure 11 Editing the Profile	24
Figure 12 Deposit Funds	25
Figure 13 Pending Transactions	26
Figure 14 Create New Item	26

Figure 15 Available Items List	26
Figure 16 My Inventory Overview	27
Figure 17 Item Detail Page	27
Figure 18 Transaction Analytics	28
Figure 19 Top Selling Items (by count)	29
Figure 20 Top Selling Items (by revenue)	30
Figure 21 Transaction History	30
Figure 22 Item List API Endpoint	31
Figure 23 POST new items via JSON	32
Figure 24 Transaction List API	32
Figure 25 Single Item Instance API	32
Figure 26 FlipFlow Admin Panel	33

1) Introduction

FlipFlow is a comprehensive web application designed to facilitate item management, financial transactions, and business analytics. Built on Django framework, it provides an interactive platform for users to track inventory, process transactions, and visualize business performance metrics. The system implements distributed architecture principles to ensure scalability and reliability across multiple nodes in an interactive and modern platform.

The system architecture supports modular development, where different Django apps handle core functionalities such as item management, market transactions, and user account operations. It is tailored for real-time responsiveness and reliability, leveraging Django's ORM for seamless database operations.

RESTful API was also used to allow for a scalable, flexible integration model that supports cross-platform communication. It makes FlipFlow extensible for partnerships with parties that already have existing inventory systems by allowing them to seamlessly integrate their products into the FlipFlow platform, reducing manual data entry and enabling automation. For example, partner companies or marketplaces can connect to FlipFlow to:

- Automatically sync their inventory
- Monitor listing status
- Modify item information via API

1.1) Key features

FlipFlow offers the following features to its clients:



- Secure accounting using Django's authentication system.
- Add, edit, or remove items to be sold, each with price, description, quantity, and images.
- Deposit & withdraw cash into your account to purchase items.
- Search for items posted by other users.
- Purchase items from others, transferring money and ownership.

- View full account info:
 - Current cash balance
 - Purchased items
 - Sold items
 - Unsold inventory
 - Inventory management dashboard
- User and item analytics, including statistics and charts.
- Robust REST API to:
 - List, create, update, and delete items
 - Handle authorized transactions
 - Enable integration with external tools
- Batch add items via JSON API (supports automation and integration)
- WSGI-threaded full DB replication every 10 seconds for backup and fault tolerance

2) Target Beneficiaries of the Project

- Primary Beneficiaries
 - **Small Business Owners:** Manage inventory and track sales performance
 - **Retail Managers:** Process transactions and generate sales reports
 - **E-commerce Operators:** Monitor product performance and customer transactions
 - **Financial Analysts:** Access transaction data for business intelligence
 - **Companies/Vendors with Existing Inventory Systems:** Through RESTful API as mentioned in Section (1): Introduction
- Secondary Beneficiaries:
 - **System Administrators:** Manage user accounts and system configuration
 - **Developers:** Extend functionality through API integrations
 - **Auditors:** Review transaction histories for compliance purposes

3) Adopted Programming Languages and Technologies

- **Primary Language and Framework:**  Python (v3.11+) and  Django (4.x)
 - Why?
 - Python with Django was chosen due to their synergy in enabling rapid, clean, and scalable web development. Python's simplicity and rich ecosystem, combined with Django's robust built-in features like an ORM, admin panel, and security tools, allow for fast prototyping and reliable production-grade applications with minimal overhead.
 - Cross-platform compatibility
 - Strong community support
 - Other Candidates
 - **Flask:** Flask is lightweight and flexible but requires manual setup for features like authentication and database handling, making it slower for full-feature development.

- **C# with ASP.NET** ASP.NET is powerful but more complex to configure and deploy. It also has a steeper learning curve and is less suited for rapid development in Python-fluent teams.
- **PHP with Laravel** Laravel is powerful but Python/Django offers better scalability, rapid development tools, and a cleaner syntax. Python also provides more flexibility for future integration with data processing and machine learning.
- Django REST Framework
- **HTML/CSS** - Frontend Technologies
- **Chart.js** - Data Visualization (for item analytics)
- **SQLite3** (Default, pluggable with PostgreSQL/MySQL) - Database
- **Bootstrap 5** - CSS Framework (for responsive frontend UI)
- Threading / Multiprocessing (for periodic DB replication)
- **Git** - version control
- **FontAwesome/Bootstrap Icons** - Icons used across the platform

4) System Architecture

4.1) Directory Structure (Simplified)



Listing 1: Directory Structure (Simplified)

Directory Structure Explanation:

- **Project/**: Contains the main Django project configuration files, including settings, URLs, and WSGI setup.c
 - **urls.py**: The main URL routing file. It includes all the routes for each individual app (e.g., **Item**, **Market**, **Accounts**). This central routing file simplifies the management of URLs and ensures that all URLs across the apps are correctly mapped.
- **Item/**: Manages all item-related operations such as creating, updating, and deleting items.
- **Market/**: Handles transactions, deposit/withdrawal functionalities, and offers management.
- **accounts/**: Manages user authentication, profile management, and transaction history.
- **api**: Exposes RESTful endpoints for all above features.
 - **api/serializers.py**: API serializers for DRF
- **static/**: Contains all the **CSS**, **JavaScript**, and image files used in the frontend.
- **templates/**: Stores all the **HTML** templates for rendering dynamic pages.
- **media/**: Uploaded item images
- **db.sqlite3**: Main database file
- **db_backups/**: Periodic DB backup storage
- **manage.py** Django management script
- **requirements.txt**: Python dependencies

4.2) Architecture Explanation

For each app (**Item**, **Market** and **accounts**), there exists mainly:

1. **models.py**
2. **views.py**
3. **templates** folder
4. **urls.py**
5. **forms.py**

We used the MVT (Model-View-Template) architectural design patterns used to separate the concerns of the app into different components:

- Model: The Model is responsible for the data and the business logic. It manages the data and defines how the data is saved or retrieved.
- View: The View is responsible for receiving user input, processing it, and returning an appropriate response (like rendering a template).
- Template: The Template in MVT is responsible for rendering the HTML or UI elements. It's a file that defines the structure and layout of the output.

For example, a typical MVT Flow is as such:

- User Request: The user sends an HTTP request (e.g., visiting a URL).
- View: The View receives the request and processes it. It interacts with the Model to fetch or modify data.
- Model Update: The Model updates the data if needed (e.g., querying the database or making business logic changes).
- Template: The View then uses the Template to render an HTML page with the updated data and returns it to the user.

this flow is illustrated by the following diagram

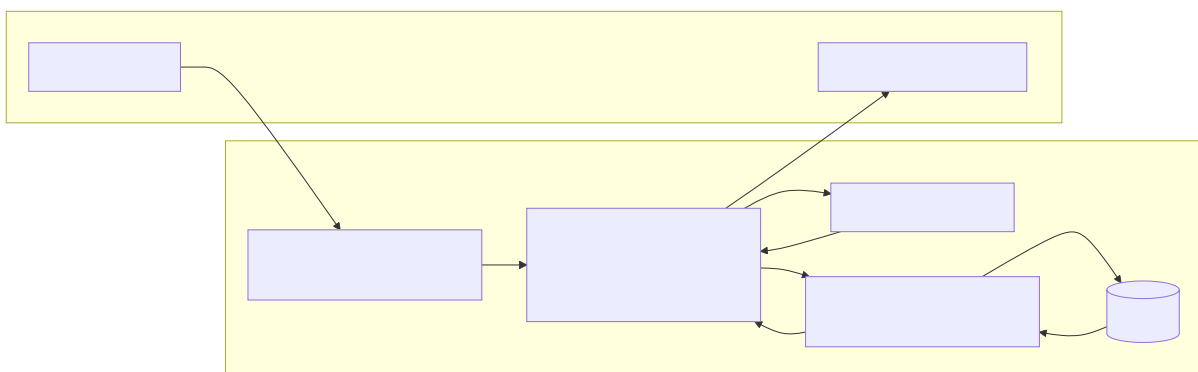


Figure 1: System Architecture Diagram

- **urls.py**: Fullfills App-specific routing. This allows each app to maintain its own URL routing while still being connected to the project. (logically it's the first step after the user's request)

- **forms.py**: Take user input and validates it. (lies between Model and View)

FlipFlow follows a client-server distributed Architecture Model. User requests are HTTP enhanced by REST. The server fetches/modifies data from the distributed SQLite database. Justification:

- Using Django's MVT pattern allows for clear separation of concerns, making the codebase maintainable and easier to extend as the project grows.
- The client-server architecture enables scalability by separating concerns: the client is responsible for the user interface, while the server handles business logic and data persistence. It also helps for scalability and security.
- The SQLite database offers robust support for data consistency and can efficiently handle growing datasets, providing a reliable backend solution.

5) Application Level Protocol

FlipFlow provides a robust API interface built with Django REST Framework (DRF).

This integration allows users and companies to interact with the system programmatically through well-structured HTTP endpoints.

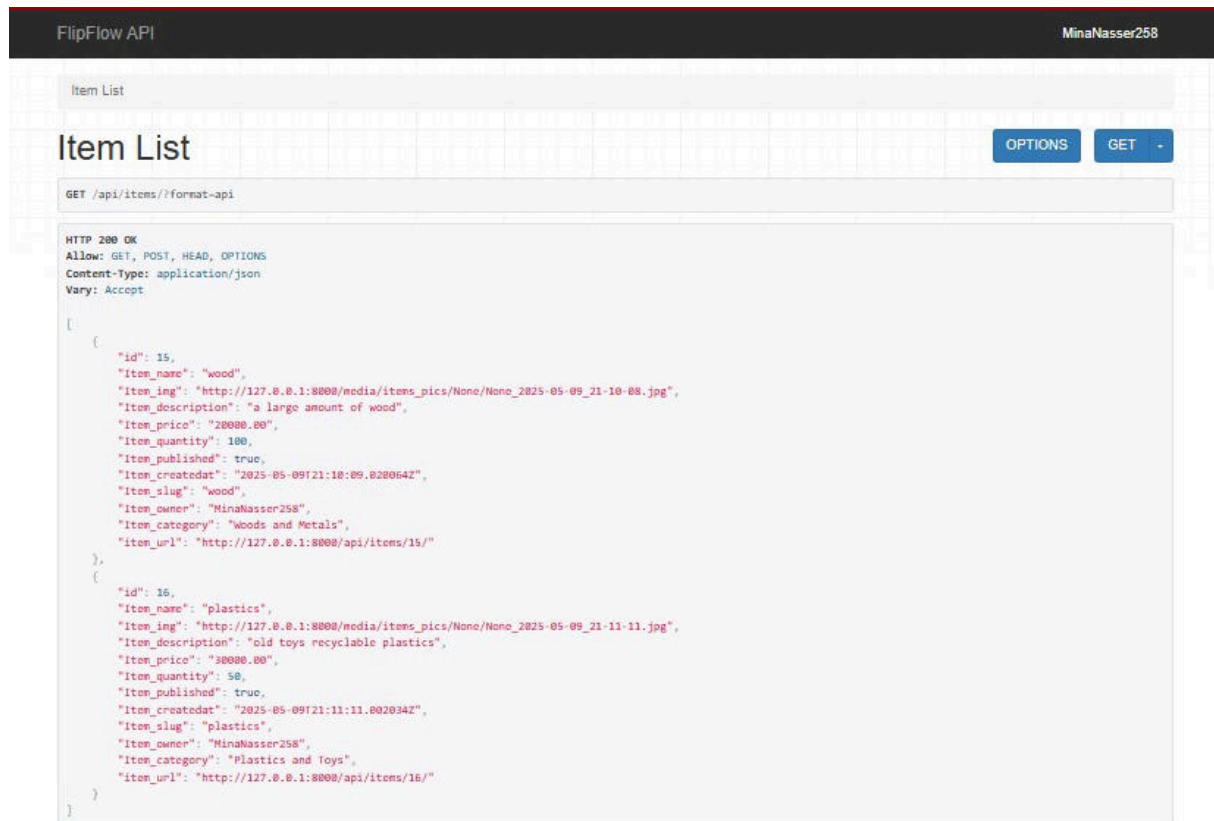
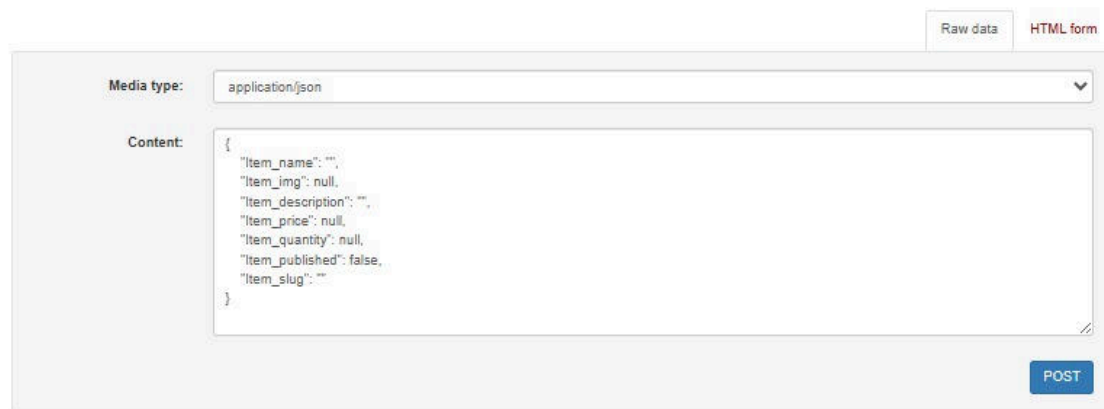


Figure 2: Item Management via API

In Figure 2, you can see the item list endpoint rendered through DRF's browsable interface. Each item includes key data such as:

- **Item_name**
- **Item_description**
- **Item_img**
- **Item_price**
- **Item_quantity**
- **Item_owner** (as a username)
- A unique **Item_url** for direct access

This interface allows users (e.g., company staff) to quickly view and test their data endpoints, making debugging and integration with external systems much easier.



The screenshot shows a web form for creating items. At the top right, there are two tabs: 'Raw data' and 'HTML form'. Below the tabs, there is a 'Media type' dropdown menu currently set to 'application/json'. Underneath, there is a 'Content' label followed by a large text area containing a JSON object:

```
{
  "Item_name": "",
  "Item_img": null,
  "Item_description": "",
  "Item_price": null,
  "Item_quantity": null,
  "Item_published": false,
  "Item_slug": ""
}
```

. At the bottom right of the form, there is a blue button labeled 'POST'.

Figure 3: Creating Items with JSON

Figure 3 demonstrates how users can POST new items via JSON using DRF's interactive interface. Supported features:

- Posting multiple items by sending bulk JSON payloads.
- Structured validation before data is saved.
- Automatic handling of image uploads via **Item_img**.

This simplifies backend workflows — a logistics manager, for example, can automate product uploads directly from an ERP system using an HTTP client.

All API operations are secured by authentication and permissions, regular users can:

- Only view and manage their own items.
- Access a read-only list of their authorized transactions.

Admins can:

- View and manage all data.
- Perform advanced tasks like bulk updates or deletion.

This ensures that sensitive data is protected and users only interact with what they're authorized to access.

Advantages of Using Django REST Framework:

- **Browsable Interface:** DRF's UI lets developers interact directly with the API from a browser.
- **Serializer-based validation:** Prevents invalid data from entering the system.
- **Scalable:** Can be connected to mobile apps, IoT systems, or other business software.
- **Maintainable:** Follows Django's clean design patterns — easy to update, expand, and secure.

Business Impact: for companies, the API integration means:

- Faster item uploads and inventory updates.
- Integration with third-party tools, like analytics dashboards or CRMs.
- Custom automation scripts can run periodic updates via the API.
- Cleaner user experience for non-technical staff using DRF's interface.

6) Distributed Database Design

FlipFlow employs a distributed database design where the database is replicated across multiple servers. This architecture enables the system to fetch data locally, enhancing performance and ensuring high availability.

6.1) Design Overview

- **Replication Across Servers:** The database is replicated across several servers, where each server holds a full copy of the database. This allows for data fetching from the nearest server, reducing latency and improving overall system response time.
- **High Availability:** In case of server failure, the system remains operational as other replicas continue to serve data. This redundancy ensures fault tolerance and prevents downtime.

6.2) Justification for the Distributed Design

- **Local Data Fetching:** The system fetches data from the nearest server replica, minimizing latency and ensuring quick access to data.
- **High Availability:** By replicating the database across multiple servers, the system guarantees high availability. If one server fails, the others continue to provide data, making the system resilient to failures.

This distributed database approach allows FlipFlow to scale seamlessly while maintaining data consistency and availability, ensuring a robust and high-performing system.

6.3) Database schema

6.3.1) Item_category

Item categories are stored here, enabling efficient retrieval across distributed servers, allowing users to view products from any replica.

```
1 CREATE TABLE IF NOT EXISTS "Item_category" (  
2     "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
3     "Category_name" VARCHAR(100) NOT NULL,  
4     "Category_slug" VARCHAR(50) NULL UNIQUE  
5 );
```

[sql](#)

6.3.2) Item_item

The core of the platform's items is stored here, including item details such as name, description, price, and quantity. All changes are replicated across other servers.

```
1 CREATE TABLE IF NOT EXISTS "Item_item" (sql
2     "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
3     "Item_name" VARCHAR(100) NOT NULL,
4     "Item_description" TEXT NOT NULL,
5     "Item_price" DECIMAL NOT NULL,
6     "Item_quantity" INTEGER NOT NULL,
7     "Item_published" BOOL NOT NULL,
8     "Item_createdat" DATETIME NOT NULL,
9     "Item_slug" VARCHAR(50) NULL UNIQUE,
10    "Item_category_id" BIGINT NULL REFERENCES "Item_category" ("id")
11    DEFERRABLE INITIALLY DEFERRED,
12    "Item_img" VARCHAR(100) NULL,
13    "Item_owner_id" INTEGER NULL REFERENCES "auth_user" ("id")
14    DEFERRABLE INITIALLY DEFERRED
15 );
```

6.3.3) accounts_profile

User profiles are stored here, ensuring that user information is consistently available across all nodes in the distributed database.

```
1 CREATE TABLE IF NOT EXISTS "accounts_profile" (sql
2     "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
3     "slug" VARCHAR(50) NULL,
4     "bio" TEXT NOT NULL,
5     "country" VARCHAR(2) NOT NULL,
6     "address" VARCHAR(100) NOT NULL,
7     "birth_date" DATE NULL,
8     "joindate" DATETIME NOT NULL,
9     "user_id" INTEGER NOT NULL UNIQUE REFERENCES "auth_user" ("id")
10    DEFERRABLE INITIALLY DEFERRED,
11    "balance" DECIMAL NOT NULL,
12    "img" VARCHAR(100) NULL
13 );
```

6.3.4) Market_transaction

Transaction data is stored here and replicated across all distributed peers, enabling seamless handling of transactions, regardless of which server is accessed.

```
1  CREATE TABLE IF NOT EXISTS "Market_transaction" (sql
2      "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
3      "transaction_type" VARCHAR(10) NOT NULL,
4      "amount" DECIMAL NOT NULL,
5      "created_at" DATETIME NOT NULL,
6      "from_approve" BOOL NOT NULL,
7      "to_approve" BOOL NOT NULL,
8      "admin_approve" BOOL NOT NULL,
9      "user_from_id" INTEGER NOT NULL REFERENCES "auth_user" ("id")
10     DEFERRABLE INITIALLY DEFERRED,
11     "user_to_id" INTEGER NOT NULL REFERENCES "auth_user" ("id")
12     DEFERRABLE INITIALLY DEFERRED,
13     "transaction_status" VARCHAR(10) NOT NULL,
14     "items_id" BIGINT NULL REFERENCES "Item_item" ("id") DEFERRABLE
15     INITIALLY DEFERRED
16 );
```

7) Time Plan

The following Gantt chart outlines the timeline of major phases in the FlipFlow development cycle.

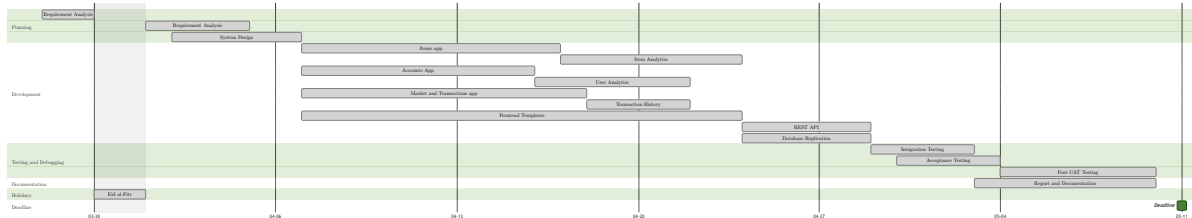


Figure 4: FlipFlow Timeline

7.1) Planning phase

This phase involved analyzing project needs and outlining a system blueprint.

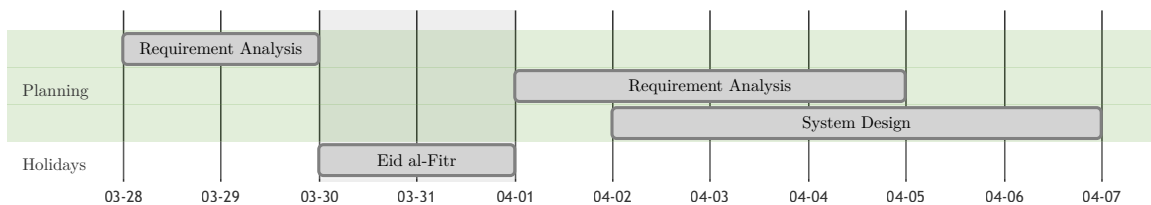


Figure 5: Planning Phase Timeline

7.2) Development Phase

This is the core implementation phase, where the major components of the system were built concurrently.

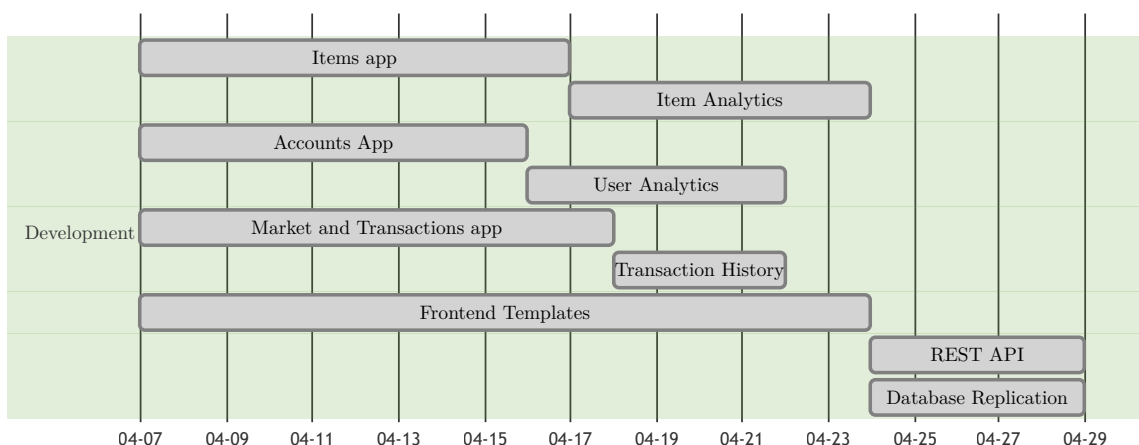


Figure 6: Development Phase Timeline

7.3) Testing Phase

This phase ensured system reliability and correctness after development.

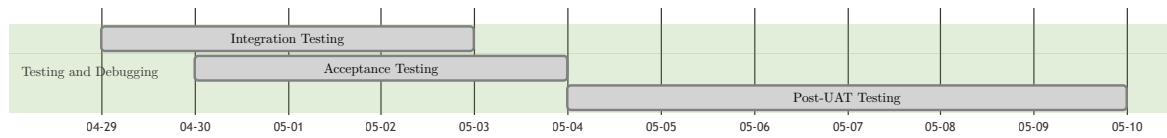


Figure 7: Testing Phase Timeline

7.4) Documentation Phase

Project report, presentation, technical documentation, and final writeups.

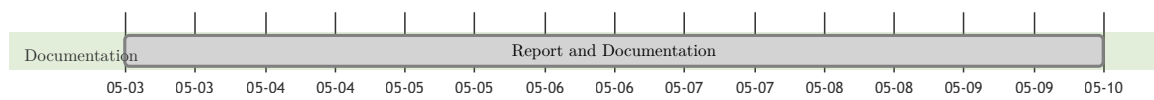


Figure 8: Documentation Phase Timeline

8) Testing

The testing approach for FlipFlow focused on both component-level and system-level validation throughout the development cycle.

8.1) Component Testing

Each module—such as the Items app, Accounts app, and Market app—was tested in isolation to ensure that individual functionalities like item creation, user registration, wallet updates, and transaction handling performed as expected. Inputs were varied to check for edge cases and error handling, such as attempting transactions with insufficient balance or submitting incomplete forms.

8.2) System Testing

Once components were integrated, system testing was performed to validate the end-to-end workflow from user signup to completing a transaction. This included navigating through the UI, performing actions like deposits, listing items, purchasing, and reviewing analytics. Cross-module interactions and data integrity were also validated—such as ensuring balance deductions matched transaction logs.

8.3) Testing Results

testing confirmed that the major functionalities work correctly and consistently across different scenarios. Known edge cases like missing input validation, API error handling, and concurrency issues were noted and addressed during debugging. The UI was tested in Chromium-based browsers to ensure consistency.

9) Needed Resources

You can view the list of languages and technologies we used in “[Section \(3\): Adopted Programming Languages and Technologies](#)”. In addition, we also utilized the following:

- Python Libraries: the `requirements.txt` file

```
1  asgiref==3.8.1
2  beautifulsoup4==4.13.3
3  Django==5.2
4  django-bootstrap4==25.1
5  django-countries==7.6.1
6  django-filter==25.1
7  djangorestframework==3.16.0
8  Markdown==3.8
9  pillow==11.1.0
10 pytz==2025.2
11 soupsieve==2.6
12 sqlparse==0.5.3
13 typing_extensions==4.13.0
14 tzdata==2025.2
```

[text](#)

- [Typst](#) for the project report and presentation
- [Mermaid](#) for drawing the project timeline Gantt chart
- [Markdown](#) for the project’s `README` file
- [Zotero](#) for managing references and research materials

10) End-User Guide

10.1) Running the Server

10.1.1) Install the requirements

🔗 Install the Requirements

```
cd flipflow
pip install FlipFlow\Project\requirements.txt
```

10.1.2) Run commands open server.py

This is an automated python script which opens the venv and runs the server for you.

```
1  import os
2  import webbrowser
3  import socket
4  hostname = socket.gethostname()
5  IPAddr = socket.gethostbyname(hostname)
6  webbrowser.open("http://{}:8000".format(IPAddr))
7  script_dir = os.path.dirname(os.path.abspath(__file__))
8
9  # Change the working directory to the script's directory
10 os.chdir(script_dir)
11 print("Current working directory:", os.getcwd())
12
13 os.system('cmd /k "cd FlipFlow & cd Scripts & activate & cd.. & cd
Project & python manage.py runserver {}:8000".format(IPAddr))
```

10.1.3) Run the Server

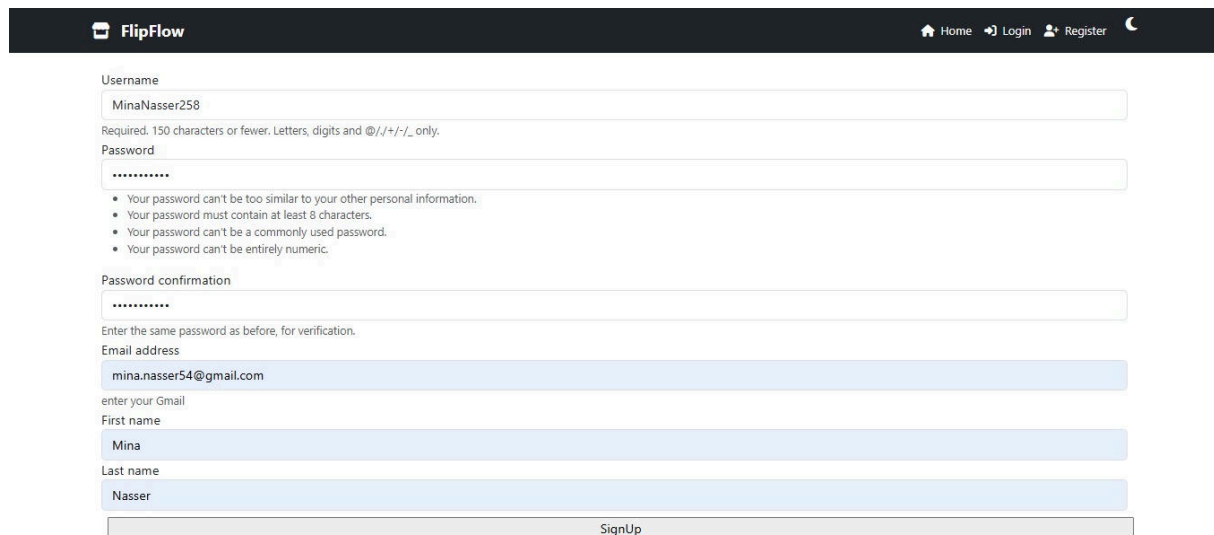
🔗 Run the Server

```
.\FlipFlow\Scripts\activate
python .\Flipflow\Project\manage.py runserver
```

Your application should now be running at <http://localhost:8000> or <http://<Your-IP-Address>:8000>.

10.2) FlipFlow Walkthrough

The following is a step-by-step explanation of how to use FlipFlow supported by screenshots.



The image shows the FlipFlow Signup page. At the top, there is a dark navigation bar with the FlipFlow logo on the left and links for Home, Login, Register, and a moon icon on the right. The main form area has a light background. It starts with a 'Username' field containing 'MinaNasser258'. Below it is a note: 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.' The 'Password' field is masked with dots. Below the password field is a list of password requirements: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.' The 'Password confirmation' field is also masked with dots, with a note 'Enter the same password as before, for verification.' Below this is the 'Email address' field containing 'mina.nasser54@gmail.com'. There is a small note 'enter your Gmail' above the 'First name' field, which contains 'Mina'. The 'Last name' field contains 'Nasser'. At the bottom of the form is a 'SignUp' button.

Figure 9: Signup page

Users begin by signing up or logging in through a clean and responsive authentication interface.

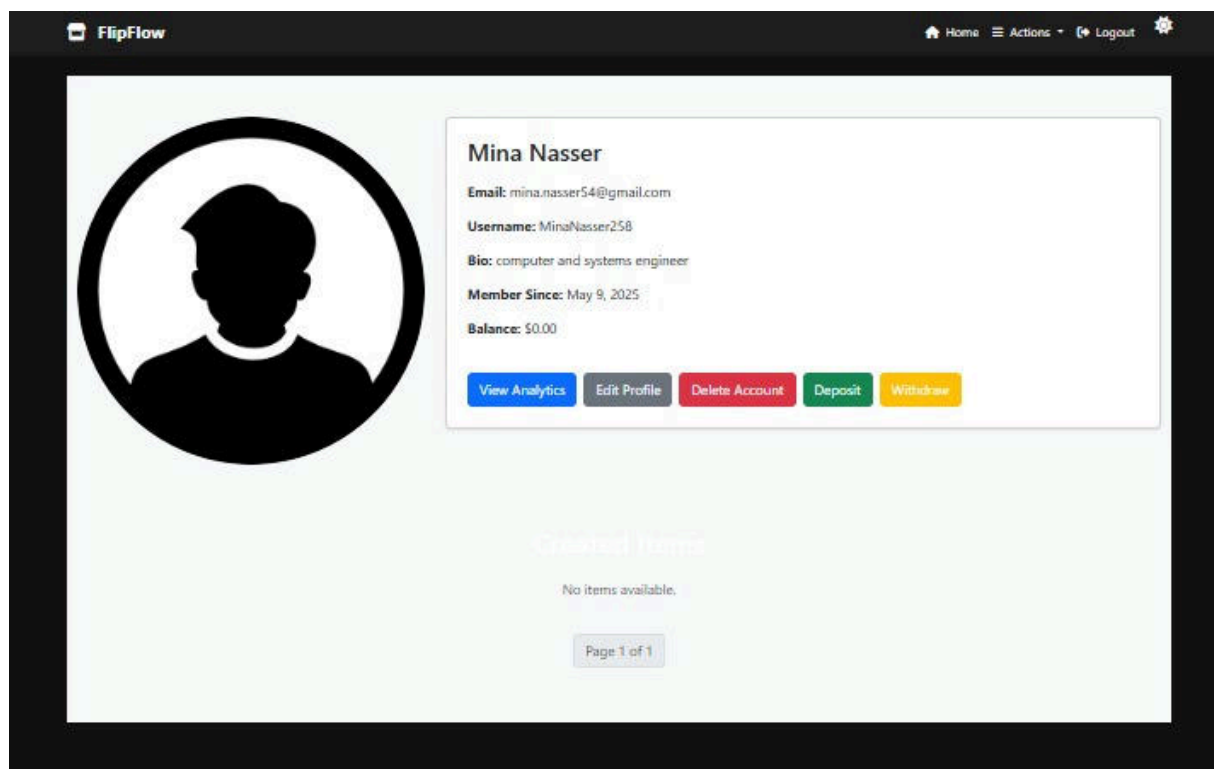
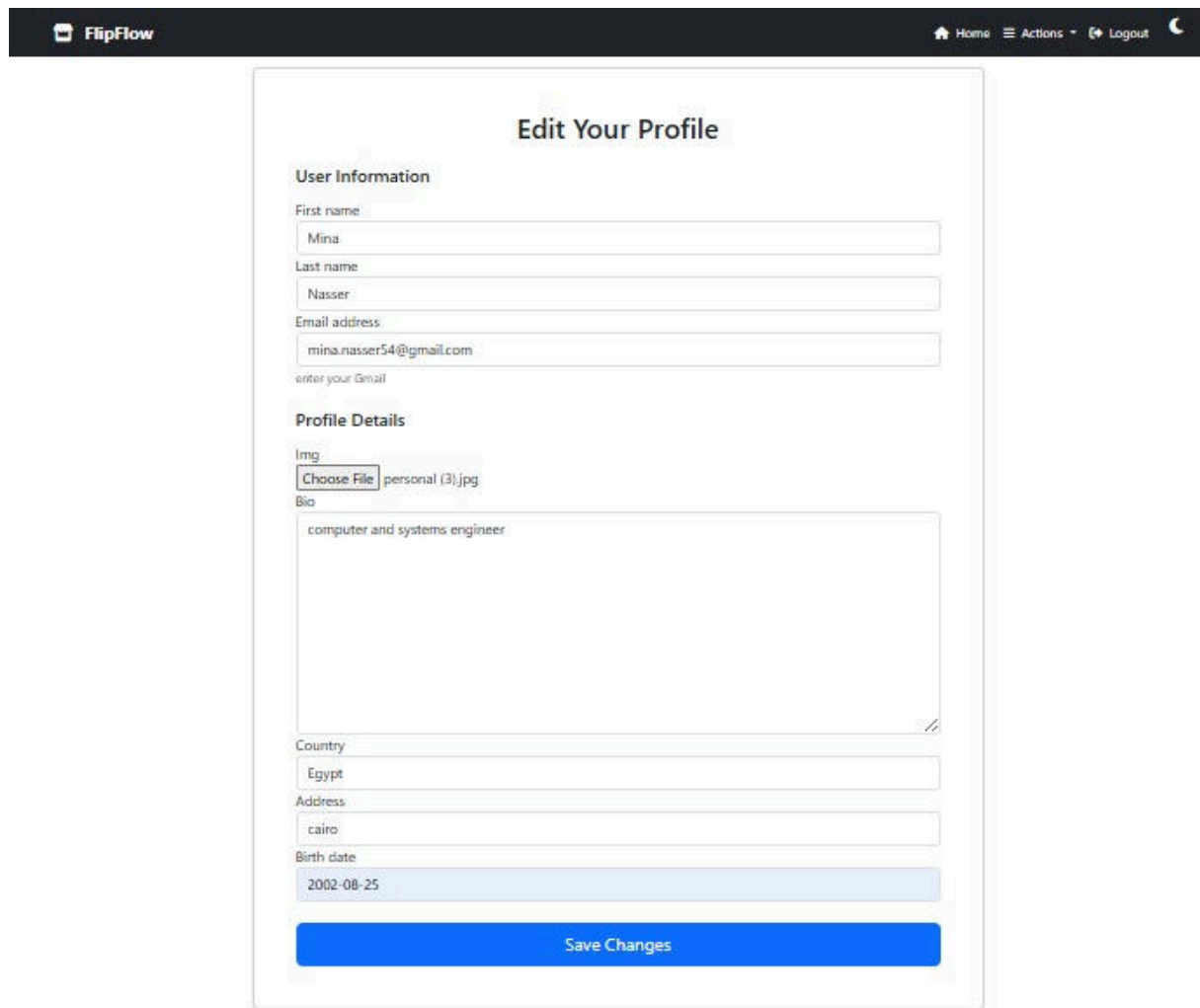


Figure 10: User Profile Page

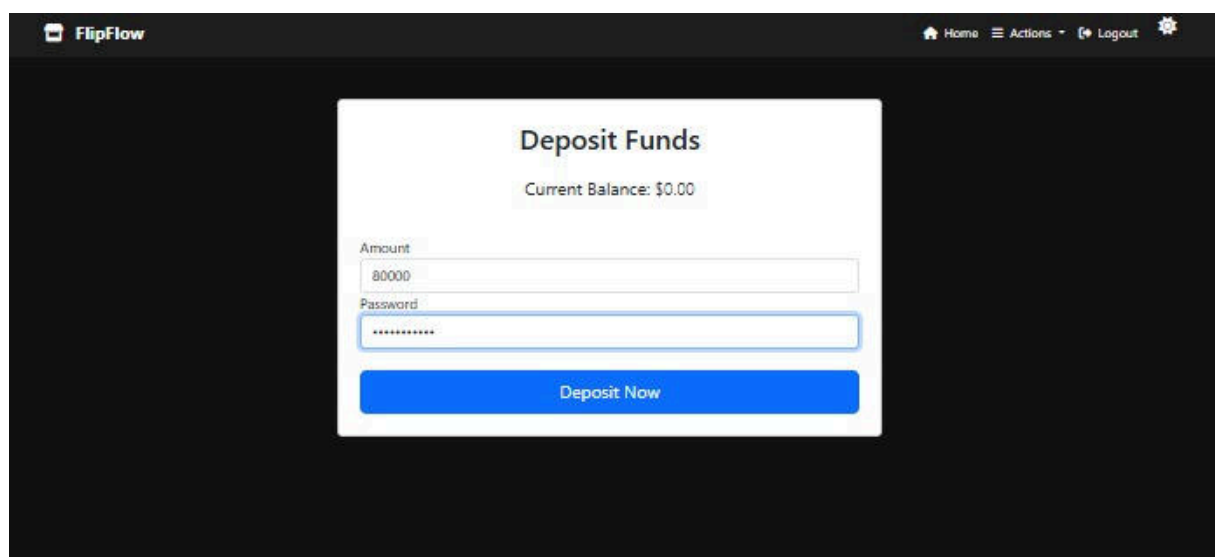
After logging in, you can see a summarized view of the user's profile, wallet balance and quick-access controls.



The screenshot shows the 'Edit Your Profile' form in the FlipFlow application. The form is titled 'Edit Your Profile' and is divided into two main sections: 'User Information' and 'Profile Details'. The 'User Information' section contains fields for 'First name' (Mina), 'Last name' (Nasser), and 'Email address' (mina.nasser54@gmail.com). The 'Profile Details' section contains fields for 'Img' (Choose File, personal (3).jpg), 'Bio' (computer and systems engineer), 'Country' (Egypt), 'Address' (cairo), and 'Birth date' (2002-08-25). A blue 'Save Changes' button is located at the bottom of the form.

Figure 11: Editing the Profile

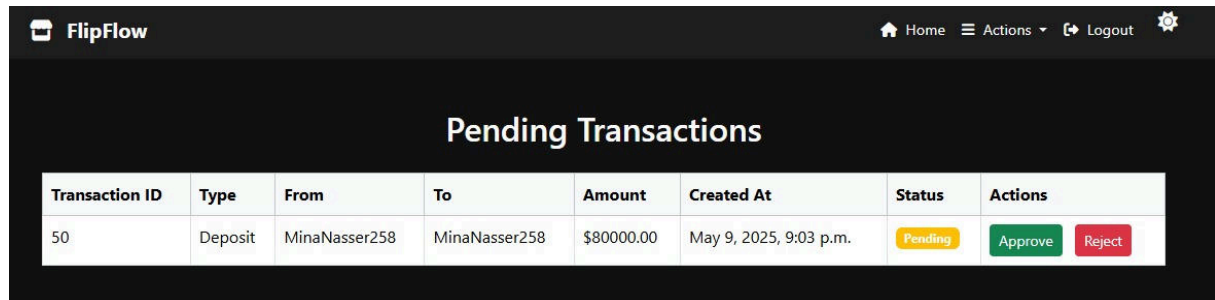
Users can edit personal details such as name, email, profile image, biography, country, address and birth date.



The screenshot shows the 'Deposit Funds' form in the FlipFlow application. The form is titled 'Deposit Funds' and displays the 'Current Balance: \$0.00'. It contains two input fields: 'Amount' (80000) and 'Password' (masked with asterisks). A blue 'Deposit Now' button is located at the bottom of the form.

Figure 12: Deposit Funds

Users can deposit funds to their FlipFlow wallet, which is essential for making transactions.

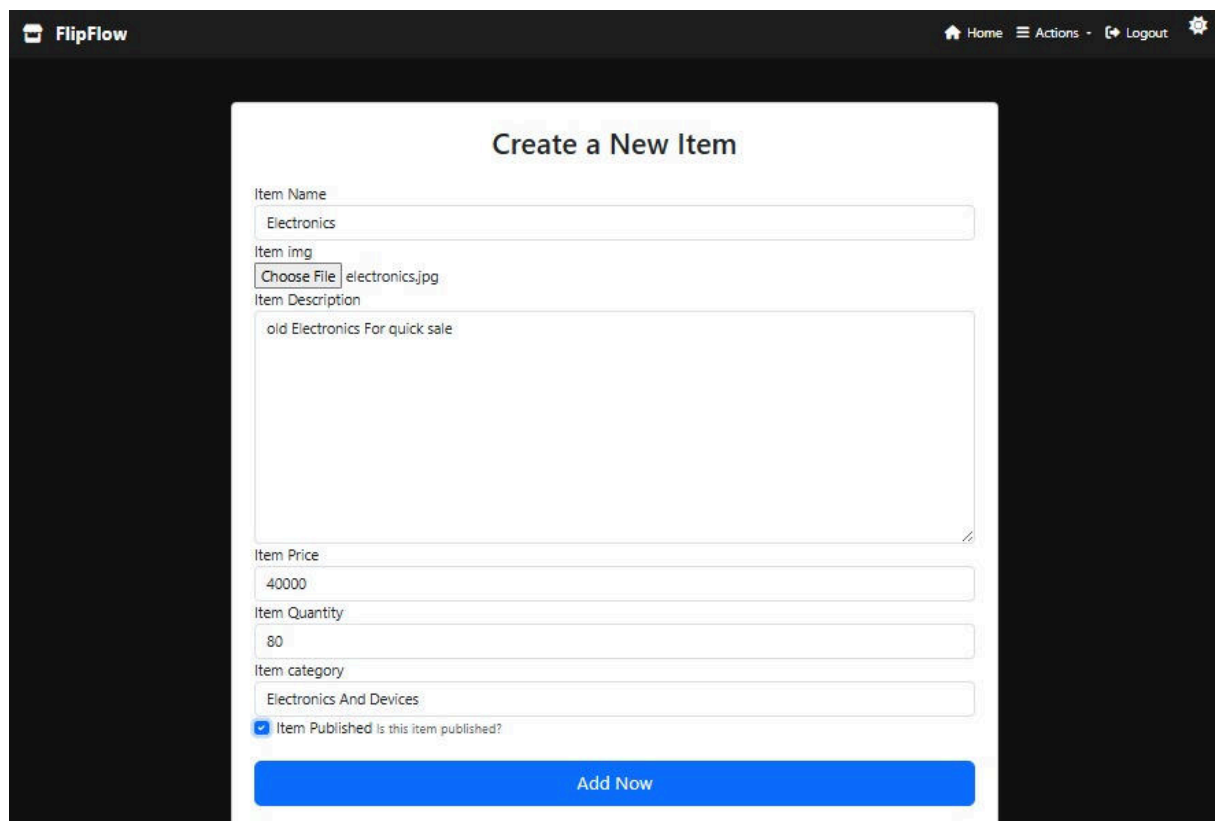


The screenshot shows the 'Pending Transactions' page in the FlipFlow application. At the top, there is a navigation bar with the FlipFlow logo, a home icon, a menu icon, 'Actions', a logout icon, and a settings icon. The main heading is 'Pending Transactions'. Below it is a table with the following data:

Transaction ID	Type	From	To	Amount	Created At	Status	Actions
50	Deposit	MinaNasser258	MinaNasser258	\$80000.00	May 9, 2025, 9:03 p.m.	Pending	<button>Approve</button> <button>Reject</button>

Figure 13: Pending Transactions

Pending transactions are tracked and managed in this interface until they are approved by the administrator.



The screenshot shows the 'Create a New Item' form in the FlipFlow application. The form is titled 'Create a New Item' and contains the following fields:

- Item Name: Electronics
- Item img: Choose File electronics.jpg
- Item Description: old Electronics For quick sale
- Item Price: 40000
- Item Quantity: 80
- Item category: Electronics And Devices
- ☒ Item Published Is this item published?

At the bottom of the form is a blue button labeled 'Add Now'.

Figure 14: Create New Item

Sellers can create new items with detailed metadata such as image, description, price, available quantity and category.

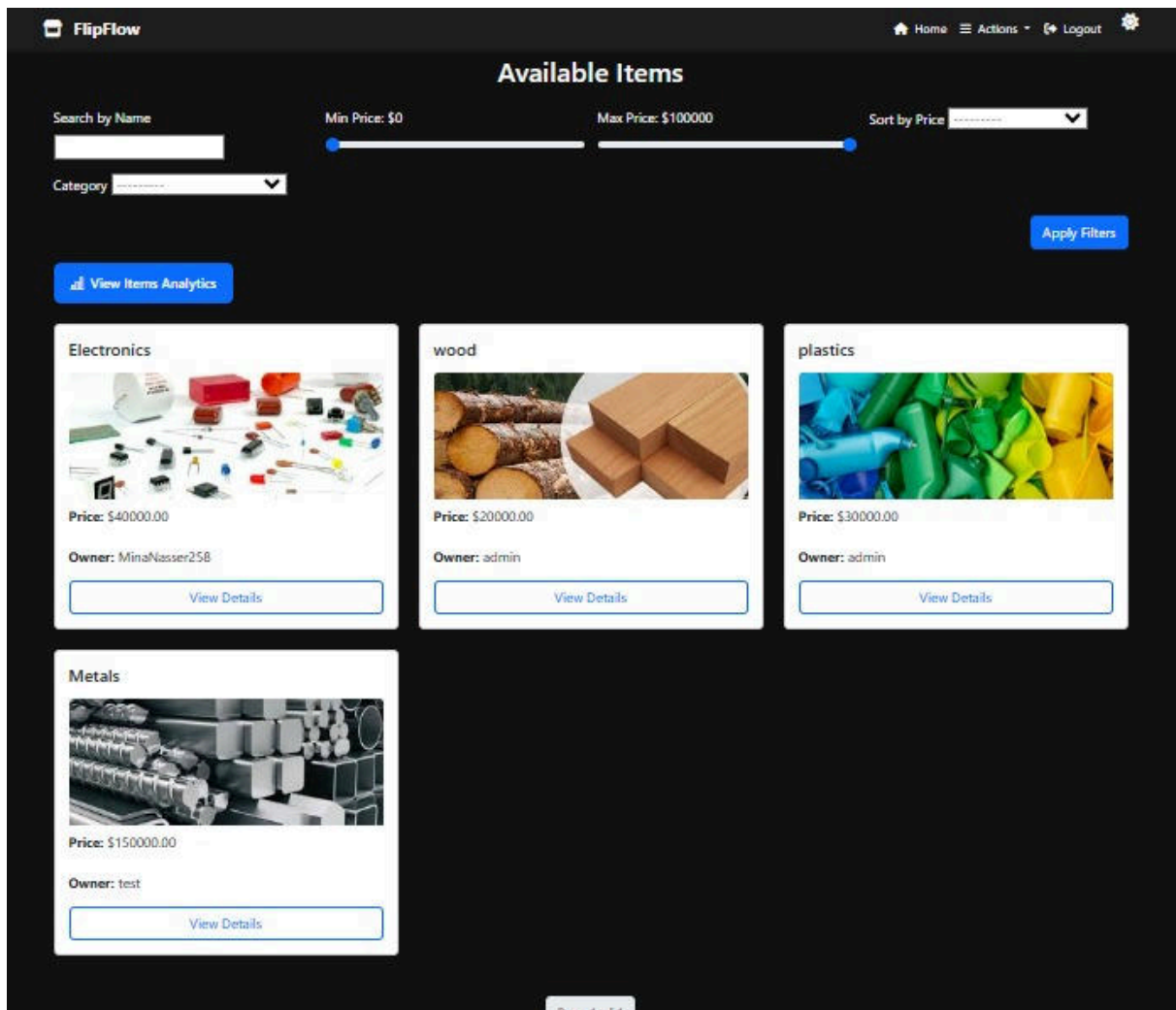
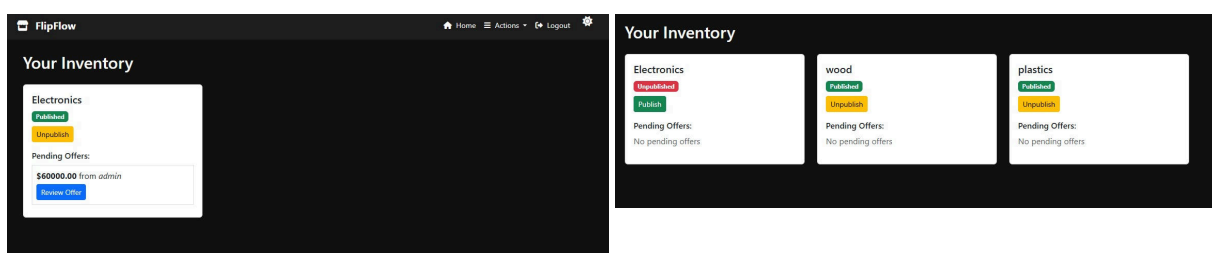


Figure 15: Available Items List

A dynamic list of all available marketplace items with filtering and sorting options.



(a) User with a pending offer for Electronics item (b) Updated Inventory of admin after the user accepted the offer from 'admin'

Figure 16: My Inventory Overview

Users can review their inventory through this page. They can also choose to publish or unpublish the items for sale and review pending offers for their items.

The screenshot shows a web interface for an item detail page. The item is titled 'Electronics' and belongs to the category 'Electronics And Devices'. It has a price of \$40000.00 and an available quantity of 80. The owner is 'admin'. The description is 'old Electronics For quick sale'. It is published and was added on May 9, 2025, at 9:09 p.m. Below the details are three buttons: 'Back to Listings' (grey), 'Edit' (green), and 'Delete' (red). Underneath is a section titled 'Ownership History' containing a table with one transaction record.

Electronics

Category: Electronics And Devices

Price: \$40000.00

Available Quantity: 80

Owner: [admin](#)

old Electronics For quick sale

Published: Yes

Added on: May 9, 2025, 9:09 p.m.

[Back to Listings](#) [Edit](#) [Delete](#)

Ownership History

Buyer	Seller	Transaction Date	Amount	Status
admin	MinaNasser258	May 9, 2025, 9:16 p.m.	\$60000.00	Completed

Figure 17: Item Detail Page

Item page with details such as item description, category, price, owner, and ownership history.



Figure 18: Transaction Analytics

User Transaction Analytics page with details such as “Total Earned (Sales)”, “Total Deposits”, “Total Spent (Purchased)” and a lot more.

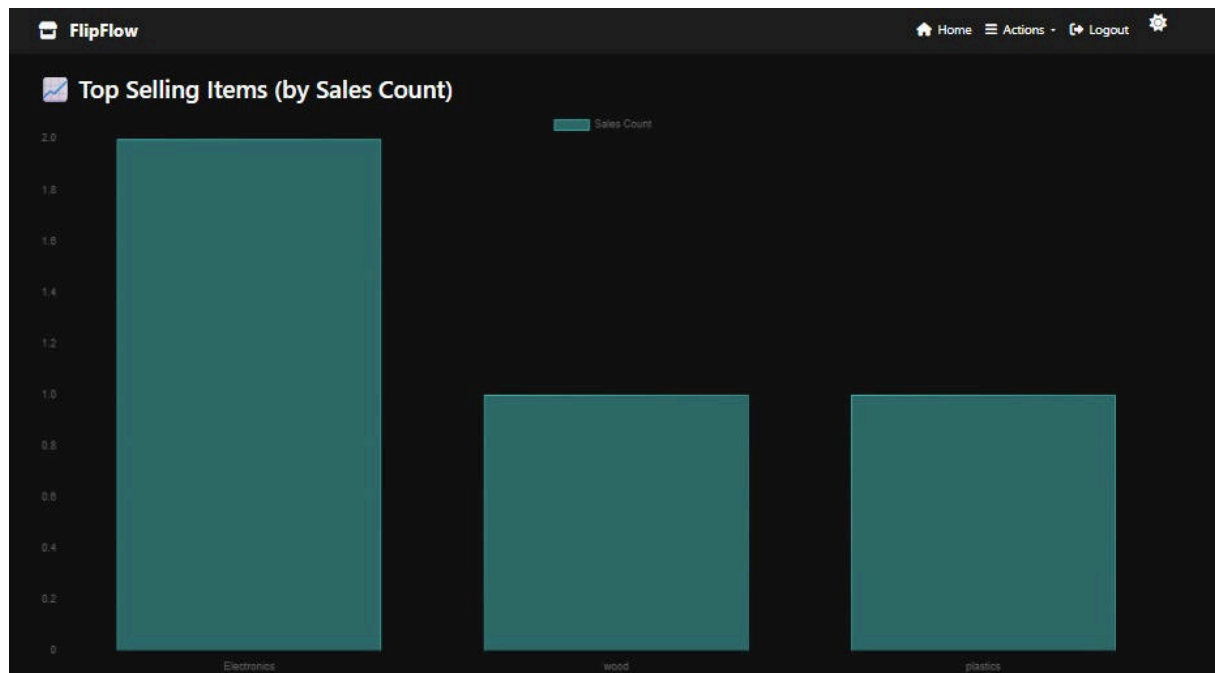


Figure 19: Top Selling Items (by count)

Figure 19 and Figure 20 both represent the “Item Analytics” page.

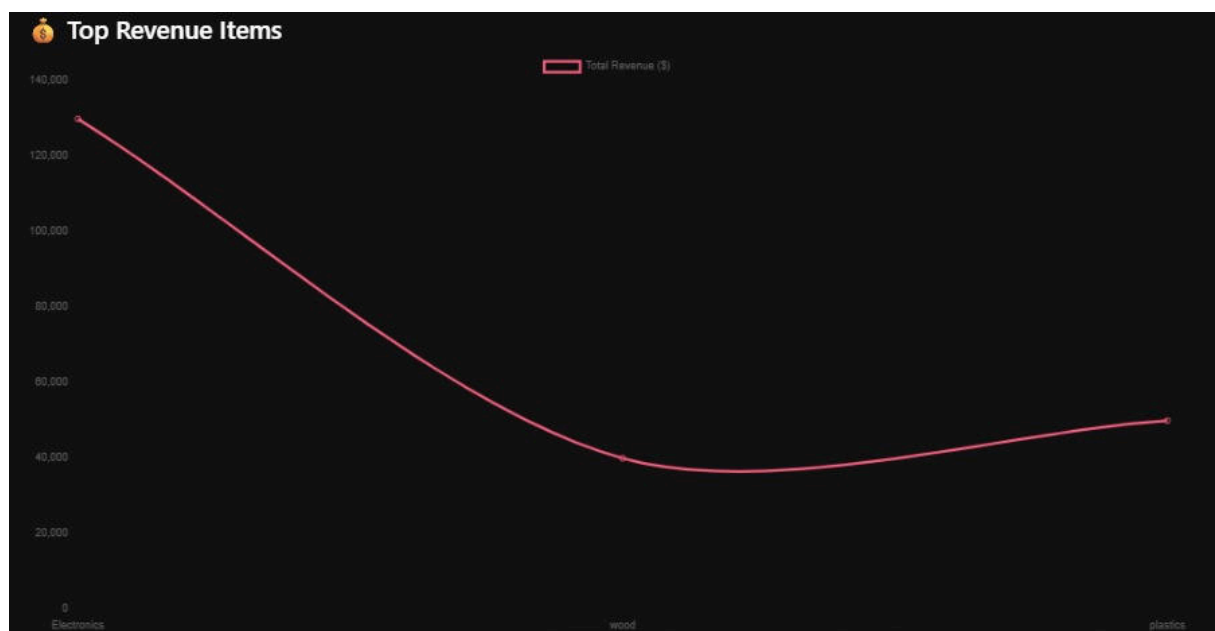


Figure 20: Top Selling Items (by revenue)

Transaction History				
Bought Items				
Item Name	Seller	Amount	Transaction Date	Status
wood	MinaNasser258	\$40000.00	May 9, 2025, 9:24 p.m.	Completed
plastics	MinaNasser258	\$50000.00	May 8, 2025, 9:20 p.m.	Completed
Sold Items				
Item Name	Buyer	Amount	Transaction Date	Status
Electronics	MinaNasser258	\$60000.00	May 9, 2025, 9:16 p.m.	Completed
Items to be Sold (Pending Offers)				
Item Name	Buyer	Offer Amount	Transaction Date	Status
No pending offers.				

Figure 21: Transaction History

Each user has access to their full transaction history including bought items, sold items and pending offers.

FlipFlow API		MinaNasser258
Item List		
Item List		
GET /api/items/?format=json		
HTTP 200 OK Allow: GET, POST, HEAD, OPTIONS Content-Type: application/json Vary: Accept		
<pre>[{ "id": 15, "item_name": "wood", "item_img": "http://127.0.0.1:8000/media/items_pics/None/None_2025-05-09_21-10-08.jpg", "item_description": "a large amount of wood", "item_price": "20000.00", "item_quantity": 100, "item_published": true, "item_createdat": "2025-05-09T21:10:09.020064Z", "item_slug": "wood", "item_owner": "MinaNasser258", "item_category": "Woods and Metals", "item_url": "http://127.0.0.1:8000/api/items/15/" }, { "id": 16, "item_name": "plastics", "item_img": "http://127.0.0.1:8000/media/items_pics/None/None_2025-05-09_21-11-11.jpg", "item_description": "old toys recyclable plastics", "item_price": "20000.00", "item_quantity": 50, "item_published": true, "item_createdat": "2025-05-09T21:11:11.002034Z", "item_slug": "plastics", "item_owner": "MinaNasser258", "item_category": "Plastics and Toys", "item_url": "http://127.0.0.1:8000/api/items/16/" }]</pre>		

Figure 22: Item List API Endpoint

Item List API endpoint. You can see more details in [Section \(5\): Application Level Protocol](#)

The screenshot shows a REST client interface with a 'Media type' dropdown set to 'application/json'. The 'Content' field contains a JSON object representing a new item. A 'POST' button is visible at the bottom right.

```
{
  "item_name": "",
  "item_img": null,
  "item_description": "",
  "item_price": null,
  "item_quantity": null,
  "item_published": false,
  "item_slug": ""
}
```

Figure 23: POST new items via JSON

POST new items via JSON. Likewise, more detail is available in [Section \(5\): Application Level Protocol](#)

The screenshot shows the 'Transaction List' API endpoint in the FlipFlow API interface. The endpoint is GET /api/transactions/. The response is an HTTP 200 OK with a JSON array of transaction objects. The interface includes a 'Transaction List' header, a 'GET /api/transactions/' endpoint, and a 'Transaction List' title. The response shows three transactions with details like id, user_from, user_to, transaction_type, transaction_status, amount, created_at, and transaction_url.

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "id": 50,
    "user_from": "MinaNasser258",
    "user_to": "MinaNasser258",
    "transaction_type": "deposit",
    "transaction_status": "completed",
    "amount": "80000.00",
    "created_at": "2025-05-04T21:03:46Z",
    "transaction_url": "http://127.0.0.1:8000/api/transactions/50/"
  },
  {
    "id": 51,
    "user_from": "test",
    "user_to": "test",
    "transaction_type": "deposit",
    "transaction_status": "completed",
    "amount": "200000.00",
    "created_at": "2025-05-05T21:14:39Z",
    "transaction_url": "http://127.0.0.1:8000/api/transactions/51/"
  },
  {
    "id": 52,
    "user_from": "admin",
    "user_to": "admin",
    "transaction_type": "deposit",
    "transaction_status": "completed",
    "amount": "300000.00",
    "created_at": "2025-05-05T21:14:39Z",
    "transaction_url": "http://127.0.0.1:8000/api/transactions/52/"
  }
]
```

Figure 24: Transaction List API

This API endpoint exposes user transactions.

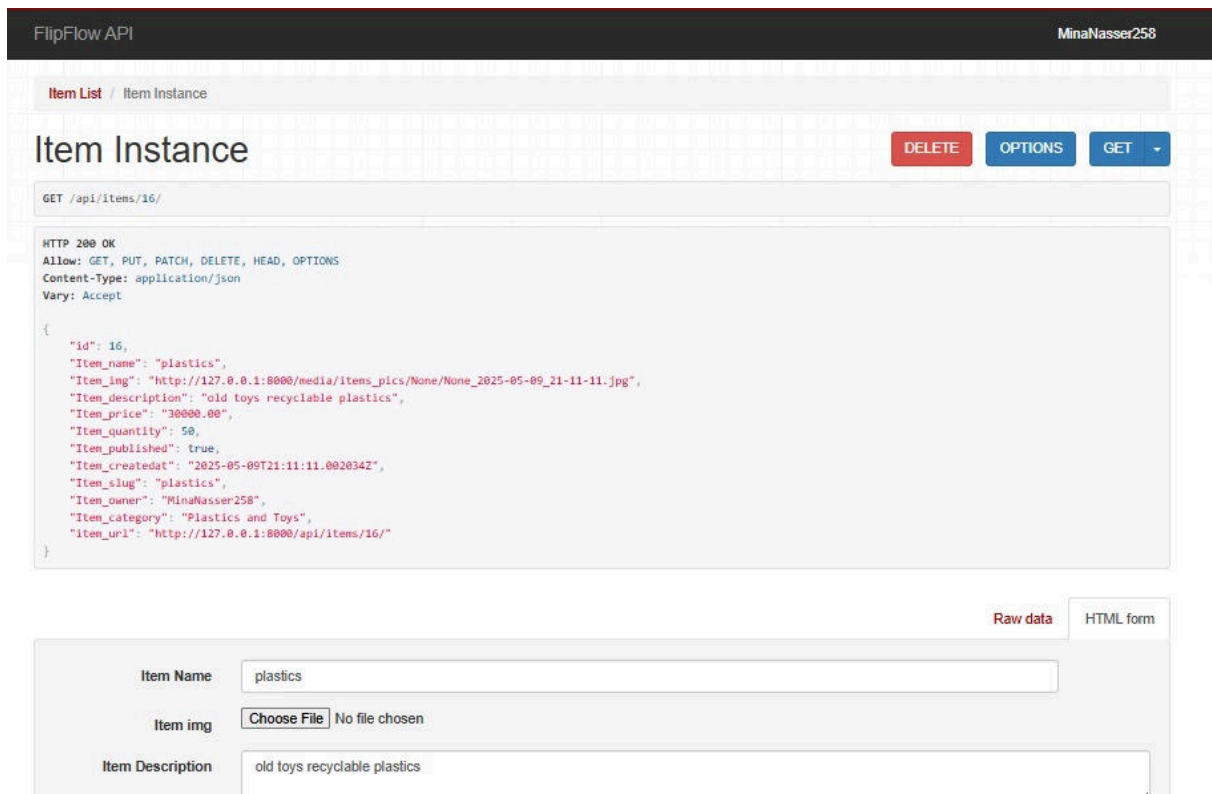


Figure 25: Single Item Instance API

An endpoint providing full details for a specific item.

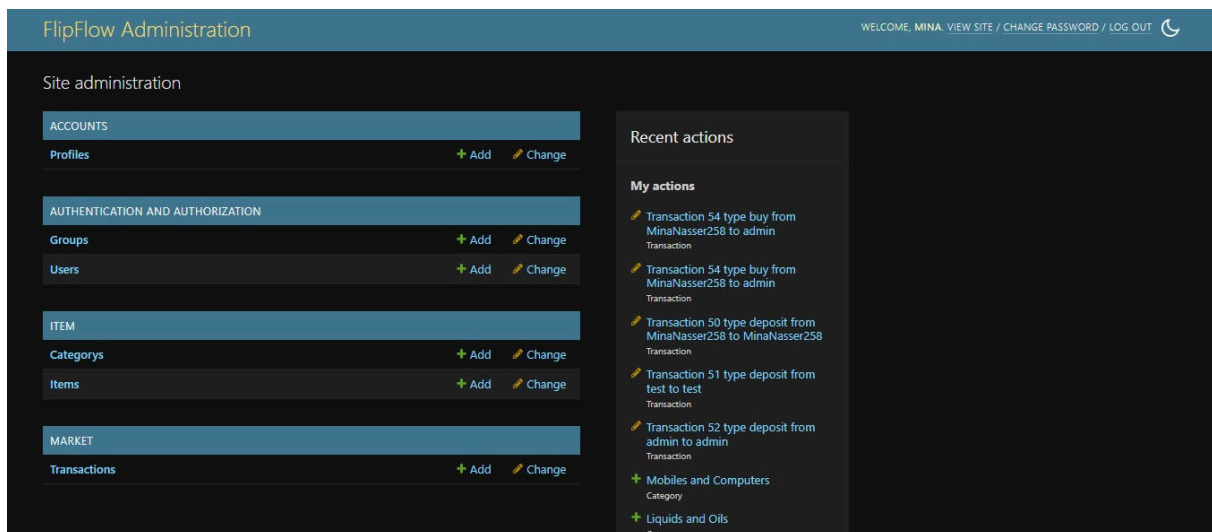


Figure 26: FlipFlow Admin Panel

FlipFlow administration panel for admins to manage user accounts, groups, permissions and transactions. They can also view recent actions log,

11) Team Member Contributions

Each team member focused on specific modules or responsibilities that aligned with their skills and the project needs. The table below outlines their individual contributions:

Name	Contribution
Mostafa Hamada Hassan Ahmed	Frontend and Transaction History
Zeyad Magdy Roushdy	REST API implementation
Mina Nasser Shehata	Items app and Version Control
Ahmed Ashraf Ahmed	Report writing and Database Replica
Mina Antony Samy Fahmy	Market and Transactions app
Mark Wagdy Wadie	Item Analytics
Mohamed Montasser Ibrahim	User analytics
George Joseph Basilious Tawadrous	Accounts app
Youssef Hatem Ahmed	Feature testing and Frontend

Table 1: Team Member Contributions

References

- [1] A. Rahman, “Comparing Web Frameworks: Flask, FastAPI, Django, NestJS, Express.js,” [Online]. Available: <https://medium.com/@arif.rahman.rhm/comparing-web-frameworks-flask-fastapi-django-nestjs-express-js-db735f1c6eba>
- [2] “What way should I choose: .NET framework or Django (Python) framework?.” [Online]. Available: <https://www.quora.com/What-way-should-I-choose-.NET-framework-or-Django-Python-framework>
- [3] “Django Project MVT Structure.” [Online]. Available: <https://www.geeksforgeeks.org/django-project-mvt-structure/>
- [4] “Difference Between MVC and MVT Architectural Design Patterns.” [Online]. Available: <https://www.geeksforgeeks.org/difference-between-mvc-and-mvt-design-patterns/>
- [5] “Django documentation | Django documentation.” [Online]. Available: <https://docs.djangoproject.com/en/5.2/>
- [6] “Django Models.” [Online]. Available: https://www.w3schools.com/django/django_models.php
- [7] “Django Forms.” [Online]. Available: <https://www.geeksforgeeks.org/django-forms/>
- [8] “Chart.js | Chart.js.” [Online]. Available: <https://www.chartjs.org/docs/latest/>
- [9] “Home - Django REST framework.” [Online]. Available: <https://www.django-rest-framework.org/>
- [10] “Mermaid Gantt Charts.” [Online]. Available: <https://mermaid.js.org/syntax/gantt.html>