

# FlipFlow

[CSE352s] Parallel and Distributed Systems Project

## Contents

1	Team Members .....	4
2	Introduction .....	9
2.1	Introduction to FlipFlow .....	10
2.2	Key Features .....	10
3	Target Beneficiaries of the Project .....	12
4	Adopted Programming Languages and Technologies .....	14
4.1	Programming Languages and technologies .....	15
5	System Architecture .....	17
5.1	Directory Structure (Simplified) .....	18
5.2	Architecture Explained .....	20
6	Application Level Protocol .....	23
7	Distributed Database Design .....	27
7.1	Design Overview .....	28
7.2	Why Distributed? .....	28

7.3 Database Schema Highlights .....	29
8 Time Plan .....	30
8.1 Time Plan .....	31
8.2 Planning phase .....	32
8.3 Development Phase .....	33
8.4 Testing Phase .....	34
8.5 Documentation Phase .....	35
9 Testing .....	36
9.1 Testing .....	37
9.1.1 Component Testing .....	37
9.1.2 System Testing .....	37
9.1.3 Results .....	37
10 Live Demo of FlipFlow .....	38
11 Team Member Contributions .....	39
References .....	41

# 1 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	Team Members .....	4
2	Introduction .....	9
2.1	Introduction to FlipFlow .....	10
2.2	Key Features .....	10
3	Target Beneficiaries of the Project .....	12
4	Adopted Programming Languages and Technologies .....	14
4.1	Programming Languages and technologies .....	15
5	System Architecture .....	17
5.1	Directory Structure (Simplified) .....	18
5.2	Architecture Explained .....	20
6	Application Level Protocol .....	23
7	Distributed Database Design .....	27
7.1	Design Overview .....	28

7.2	Why Distributed? .....	28
7.3	Database Schema Highlights .....	29
8	Time Plan .....	30
8.1	Time Plan .....	31
8.2	Planning phase .....	32
8.3	Development Phase .....	33
8.4	Testing Phase .....	34
8.5	Documentation Phase .....	35
9	Testing .....	36
9.1	Testing .....	37
9.1.1	Component Testing .....	37
9.1.2	System Testing .....	37
9.1.3	Results .....	37
10	Live Demo of FlipFlow .....	38

11 Team Member Contributions .....	39
References .....	41



## 2 Introduction

## 2.1 Introduction to FlipFlow

**FlipFlow** is a comprehensive web application built on the Django framework, designed to manage items, handle financial transactions, and provide business analytics. It uses a distributed architecture for scalability and reliability across multiple nodes.

## 2.2 Key Features

- **Item management:** Add/edit/remove items with metadata (price, description, quantity, images).
- **Financial operations:** Deposit/withdraw funds for item purchases.
- **Marketplace features:**
  - Search and purchase items from other users
  - Automatic money and ownership transfer
- **User account dashboard:**
  - View current cash balance



- Track purchased, sold, and unsold items
- Manage inventory via dashboard
- **Secure user accounts** with Django's authentication system.
- **Analytics:**
  - Item and user statistics
  - Data visualizations and charts
- **REST API capabilities**
  - External system integration
  - Batch item uploads via JSON
- **Reliability:**
  - WSGI-threaded database replication every 10 seconds for backup and fault tolerance

## 3 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
- Secondary Beneficiaries:
  - **System Administrators:** Manage user accounts and system configuration
  - **Developers:** Extend functionality through API integrations
  - **Auditors:** Review transaction histories for compliance purposes

## 4 Adopted Programming Languages and Technologies

## 4.1 Programming Languages and technologies

- **Primary Language and Framework:**  Python (v3.11+) and  Django (4.x)
  - Why?
    - Seamless synergy enabling rapid, clean, and scalable web development
    - Python offers simplicity and a rich ecosystem
    - Django includes powerful built-in tools:
      - ORM
      - Admin panel
      - Security features
    - Cross-platform compatibility
    - Strong community support
  - Other Candidates
    - Flask: Lightweight and flexible, but lacks built-in features for full-scale development (e.g., authentication, ORM).

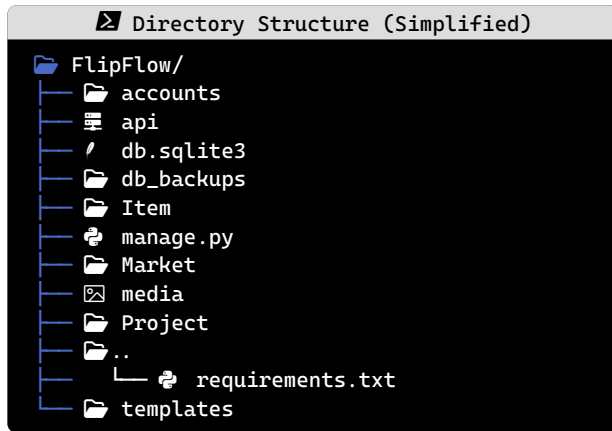
- C# with ASP.NET: Powerful but complex to configure/deploy; steep learning curve.
- PHP with Laravel: Feature-rich, but Django offers better scalability, cleaner syntax, and more flexibility for future ML/data processing integration.
- Django REST Framework
- **HTML/CSS** - Frontend Technologies
- **Chart.js** - Data Visualization (for item analytics)
- **SQLite3** (Default, pluggable with PostgreSQL/MySQL) - Database
- **Bootstrap 5** - CSS Framework (for responsive frontend UI)
- **Git** - version control
- **FontAwesome/Bootstrap Icons** - Icons used across the platform



## 5 System Architecture

## 5.1 Directory Structure (Simplified)

- **Project/** – Main Django configuration directory
- **Item/** – Handles all item operations (create, update, delete)
- **Market/** – Manages financial transactions (deposit, withdrawal) and offer logic
- **accounts/** – Responsible for user authentication, profiles, and transaction history
- **api/** – Provides RESTful endpoints for core features

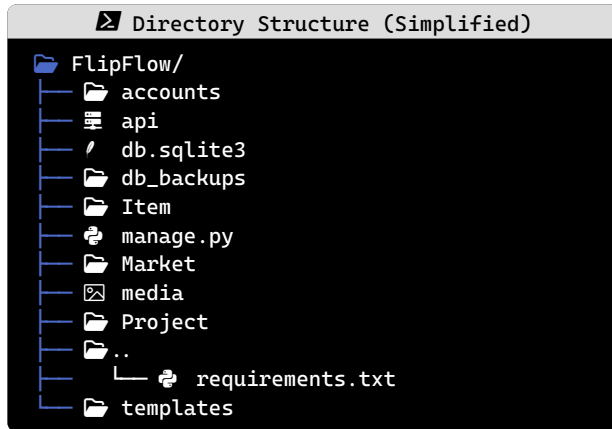


Listing 1: Directory Structure (Simplified)

# Directory Structure (Simplified) (ii)

19 / 43

- **templates/** – Contains HTML templates for dynamic page rendering
- **media/** – Holds user-uploaded item images
- **db.sqlite3** – Main database file (default)
- **db\_backups/** – Stores periodic database backups for fault tolerance
- **manage.py** – Django's command-line utility for app management
- **requirements.txt** – Lists all required Python packages



## 5.2 Architecture Explained

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) pattern to separate app concerns:

- **Model:** Handles data and business logic
- **View:** Processes user input, calls Model, returns response
- **Template:** Renders UI with data

## Typical Flow

1. User sends HTTP request
2. View processes request, interacts with Model
3. Model updates data
4. View renders Template and returns response

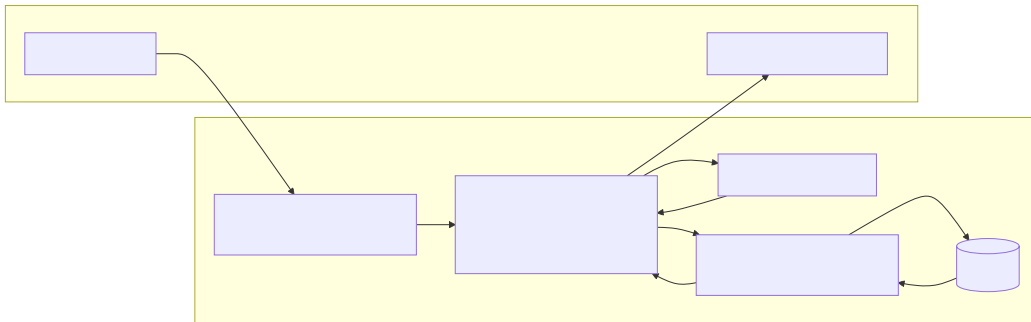


Figure 2: System Architecture Diagram

- `urls.py`: Handles app-specific routing
- `forms.py`: Validates user input (links Model & View)

FlipFlow uses a client-server architecture over REST:

- MVT ensures maintainable, scalable code
- Client handles UI; server manages logic & persistence
- SQLite ensures data consistency and scales well

## 6 Application Level Protocol

FlipFlow offers a robust API via Django REST Framework (DRF), enabling programmatic access through clean HTTP endpoints.

Figure 3 shows DRF's browsable API:

- View items: name, description, image, price, quantity, owner, URL
- Easy debugging and testing for users and staff

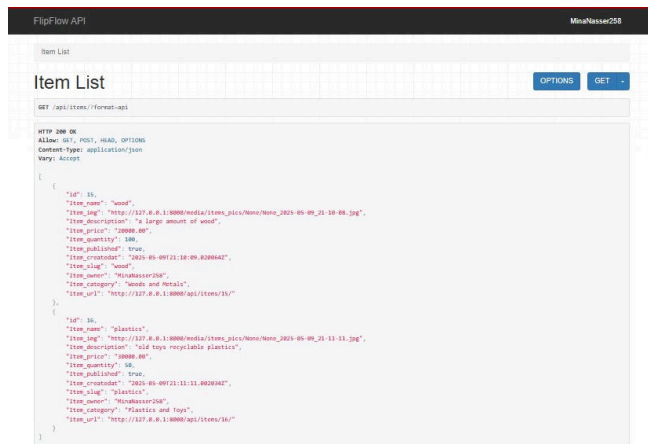


Figure 3: Item Management via API

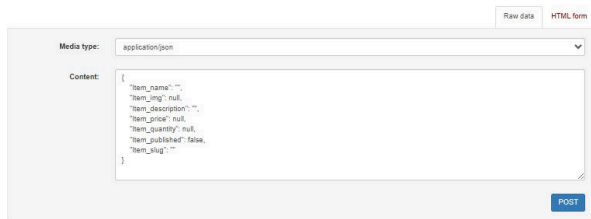


Figure 4 shows item creation via JSON:

- Supports bulk posting and validation
- Handles image uploads
- Ideal for ERP integration and automation
- Users: view/manage own items, view transactions
- Admins: full access, bulk actions
- Enforced via authentication and permissions

## Why DRF?

- Browsable UI for easy testing



The screenshot shows a web form for creating an item. At the top right, there are two tabs: 'Raw data' (selected) and 'HTML form'. Below the tabs, there is a 'Media type:' dropdown menu set to 'application/json'. Underneath, there is a 'Content:' label followed by a 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 text area is a blue 'POST' button.

Figure 4: Creating Items with JSON

- Serializer validation for data integrity
- Scalable & maintainable architecture
- Works with mobile apps, IoT, CRMs

## Business Value

- Faster inventory workflows
- Third-party tool integration
- Automation-ready API for operations
- Friendly UI for non-tech users

## 7 Distributed Database Design

FlipFlow uses a distributed database design with replicated servers for better performance and availability.

## 7.1 Design Overview

- Replication: Full database copies on multiple servers allow local data access, reducing latency.
- High Availability: If one server fails, others continue serving data — ensuring uptime and fault tolerance.

## 7.2 Why Distributed?

- Fast Access: Data is fetched from the nearest replica.
- Resilience: The system stays operational even during server failures.

This design ensures FlipFlow is scalable, consistent, and high-performing.

## 7.3 Database Schema Highlights

- **Item\_category**: Stores item categories; available across all replicas.
- **Item\_item**: Core product info (name, description, price, quantity) is fully replicated.
- **accounts\_profile**: User profiles are consistent across nodes.
- **Market\_transaction**: All transactions are synchronized for seamless access anywhere.

# 8 Time Plan

## 8.1 Time Plan

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

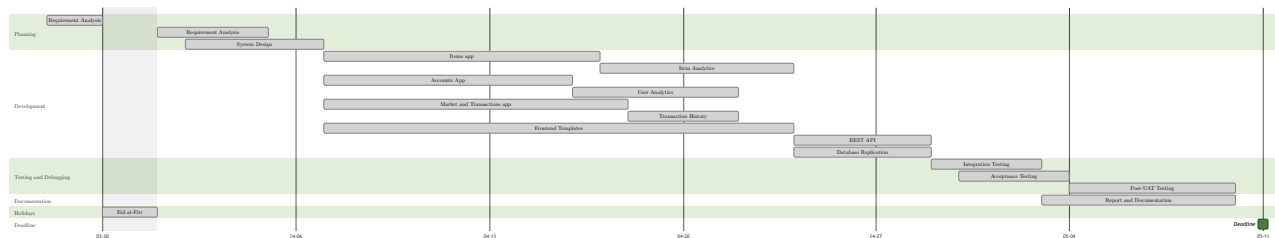


Figure 5: FlipFlow Timeline

## 8.2 Planning phase

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

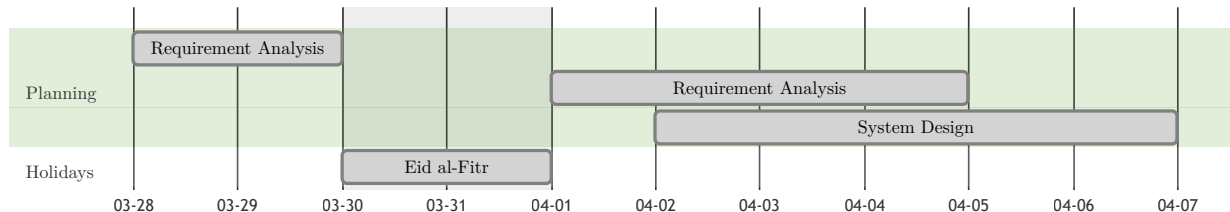


Figure 6: Planning Phase Timeline



## 8.3 Development Phase

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

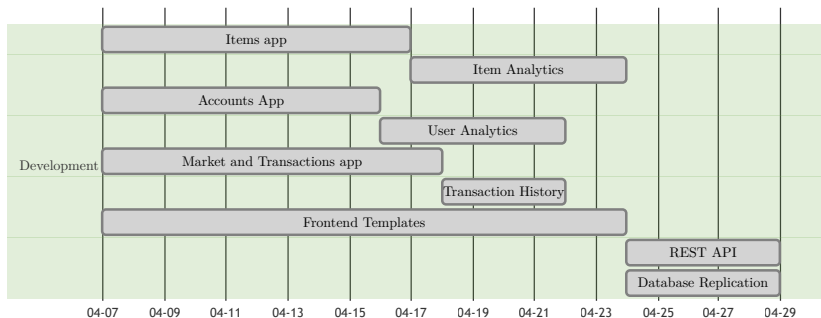


Figure 7: Development Phase Timeline

## 8.4 Testing Phase

This phase ensured system reliability and correctness after development.

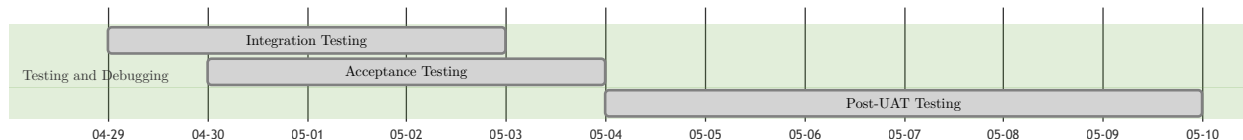


Figure 8: Testing Phase Timeline

## 8.5 Documentation Phase

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



Figure 9: Documentation Phase Timeline

## 9 Testing

## 9.1 Testing

### 9.1.1 Component Testing

- Modules (Items, Accounts, Market) tested in isolation
- Checked edge cases: incomplete forms, insufficient balance, etc.

### 9.1.2 System Testing

- Full workflow tested: signup → transaction
- Verified UI actions, cross-module logic, and data integrity

### 9.1.3 Results

- Core features worked reliably across scenarios
- Edge cases addressed; UI verified on Chromium browsers

## 10 Live Demo of FlipFlow

## 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](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>