Sample Technical Design Document (TDD) for Flipkart Application Testing

1. Introduction

1.1 Purpose

The purpose of this Technical Design Document is to outline the system architecture and technical implementation details for building an e-commerce platform similar to Flipkart. This document is intended for the development team and will provide the necessary technical guidance to implement the solution efficiently.

1.2 Scope

This document covers the architecture, components, data flow, database design, APIs, and system integration requirements. It does not include the business or functional requirements, which are covered in the BRD and FRS.

1.3 Background

Flipkart is an e-commerce platform that allows users to purchase products online. This document will detail the technical aspects required to build a scalable, secure, and high-performance platform capable of handling millions of users and transactions.

2. System Architecture

2.1 High-Level Architecture

The system is based on a three-tier architecture:

- Presentation Layer: The front-end interface (web and mobile apps) that interacts with users.

- Application Layer: The middle tier that handles the business logic.

- Data Layer: The back-end systems responsible for data storage, including databases and caches.

[Architecture Diagram] (Insert architecture diagram here showing the three-tier setup)

2.2 Key Components

1. Front-End (Web & Mobile)

   - React.js (for web interface)

   - React Native (for mobile apps)

2. Back-End

   - Node.js/Express.js (RESTful API development)

   - Microservices architecture

3. Database

   - MySQL/PostgreSQL (for transactional data)

   - Redis (for caching)

4. Payment Gateway Integration

   - Integration with third-party payment services like Paytm, Razorpay, and credit/debit cards.

5. Authentication & Authorization

   - OAuth 2.0 (for login with Google/Facebook)

   - JWT tokens (for session management)

3. Data Flow Diagram

3.1 User Registration and Login Flow

- Step 1: User accesses the registration/login page.

- Step 2: User submits credentials (email/phone number or social login).

- Step 3: The server validates the credentials using the OAuth/JWT service.

- Step 4: A session token is generated and returned to the user.

- Step 5: User is redirected to the dashboard with a valid session.

(Insert Data Flow Diagram illustrating this process)

3.2 Order Placement Flow

- Step 1: User adds items to the cart.

- Step 2: User clicks on "Proceed to Checkout."

- Step 3: The front-end sends the order request to the back-end API.

- Step 4: The back-end validates the order, confirms the inventory, and forwards the payment request to the payment gateway.

- Step 5: Upon payment success, the order is saved in the database and an email confirmation is sent to the user.

(Insert Data Flow Diagram for order placement)

4. Database Design

4.1 Entity-Relationship Diagram (ERD)

- Users Table: Stores user details such as user ID, name, email, password, and address.

- Products Table: Stores product information such as product ID, name, category, price, stock, and description.

- Orders Table: Stores order details including order ID, user ID, product ID, order status, and payment details.

- Payments Table: Stores payment-related information, such as payment ID, order ID, payment method, status, and amount.

(Insert ERD showing relationships between entities)

4.2 Database Tables

Users:

Field Name	Type	Description
user_id	INT	Primary key
name	VARCHAR(50)	Full name of the user
email	VARCHAR(100)	Email address of user
password	VARCHAR(255)	Encrypted password
address	TEXT	Delivery address

Products:

Field Name	Type	Description
product _id	INT	Primary key
name	VARCHAR(100)	Name of the product
category	VARCHAR(50)	Payment category
price	DECIMAL(10,2)	Price of the product
stock	INT	Available stock

Orders:

Field Name	Type	Description
order_id	INT	Primary key
user _id	INT	Foreign key (references Users)
status	VARCHAR(20)	Order status (Pending / Complete)
total_price	DECIMAL(10,2)	Total price of the order

Payments:

Field Name	Type	Description
payment_id	INT	Primary key
order_id	INT	Foreign key (references Orders)
method	VARCHAR(20)	Payment method (Credit Card, Paytm, etc.)
status	VARCHAR(20)	Payment status (Success/Failed)
amount	DECIMAL(10,2)	Payment amount

5. API Design

5.1 User API

- POST /api/register: Register a new user.

- POST /api/login: Log in with email/password or social media credentials.

- GET /api/user/{id}: Fetch user profile details.

5.2 Product API

- GET /api/products: Fetch list of products.

- GET /api/products/{id}: Fetch details of a specific product.

- POST /api/products: Add a new product (for sellers).

5.3 Order API

- POST /api/orders: Create a new order.

- GET /api/orders/{id}: Fetch details of a specific order.

- PUT /api/orders/{id}/status: Update order status.

5.4 Payment API

- POST /api/payments: Process a new payment.

- GET /api/payments/{id}: Fetch payment details.

6. Security Considerations

6.1 Authentication & Authorization

- OAuth 2.0 is used for social logins (Google/Facebook).

- JWT tokens are used for session management and API authentication.

6.2 Data Encryption

- All sensitive user data, including passwords, will be encrypted using AES-256.

- Payment information will be transmitted using HTTPS and will comply with PCI-DSS standards.

6.3 Access Control

- Role-based access control (RBAC) will be implemented to ensure proper permission levels (e.g., admin, seller, customer).

7. Performance Optimization

7.1 Caching

- Redis will be used for caching frequently accessed data, such as product listings and search results.

7.2 Load Balancing

- The system will implement load balancing using tools like Nginx or AWS Elastic Load Balancing to distribute traffic evenly across servers.

8. Error Handling & Logging

8.1 Error Logging

- Application logs will capture critical errors using a logging framework (e.g., Winston for Node.js).

- A separate database table will track API errors with timestamps and error codes for debugging.

8.2 Monitoring

- Monitoring tools like Prometheus or AWS CloudWatch will be used to monitor application performance, uptime, and resource usage.

9. Conclusion

This Technical Design Document outlines the architecture, data flow, database design, API specifications, and security considerations for building an e-commerce platform like Flipkart. The system is designed to be scalable, secure, and performance-optimized to meet the needs of millions of concurrent users and transactions.