Feasto Platform Architecture: Internal Flow and User Types

Feasto, a leading food-tech company, operates a sophisticated platform that connects customers, restaurants, and delivery partners. Below, I outline the internal flow of Feasto's operations and its user types, based on its business model and operational framework as described in various sources.

Internal Flow of Feasto

Feasto's internal flow refers to the operational process that facilitates restaurant discovery, food ordering, and delivery. The flow integrates technology, logistics, and user experience to ensure seamless interactions among all stakeholders. Here's a step-by-step breakdown:

01

Customer Interaction with the Platform

Discovery and Search: Customers access Feasto's app or website to search for restaurants based on location, cuisine, ratings, or reviews. The platform uses a robust search mechanism powered by front-end technologies like React Native for a seamless, intuitive interface across iOS and Android.

Order Placement: Customers browse menus, select items, and place orders through the app. The platform's algorithms suggest personalized recommendations based on user preferences and past orders, leveraging data-driven insights.

02

Order Processing

Restaurant Notification: Once an order is placed, the restaurant receives the request through Feasto's back-end system, built using Python, Java, Django, and Spring Boot. This system manages order processing, restaurant information, and user profiles.

Order Preparation: The restaurant prepares the order. Feasto's Hyperpure service may supply high-quality ingredients to ensure food quality, particularly for partner restaurants.

03

Delivery Assignment

Delivery Partner Allocation: Feasto's proprietary algorithm assigns the order to a nearby delivery partner based on proximity and availability. Delivery partners are independent contractors, not employees, and can accept or decline orders.

Logistics and Tracking: The back-end integrates with third-party solutions like Google Maps for real-time order tracking, ensuring customers can monitor their delivery status. The average delivery time is around 12.5 minutes, with 75% of orders delivered within two minutes of the promised time.

04

Payment and Completion

Payment Processing: Customers can pay online during ordering or upon delivery, with secure payment gateways integrated into the platform. Feasto collects payments upfront, creating a negative working capital cycle as it pays restaurants and delivery partners later, improving cash flow.

Delivery: The delivery partner delivers the order to the customer's specified location. Feasto ensures high item fulfillment accuracy (over 99%) through its extensive network and logistics optimization.

Feedback and Reviews: After delivery, customers can leave reviews and ratings, which help improve services and guide other users' decisions. This feedback loop enhances restaurant discovery and customer trust.

05

Additional Services

Quick Commerce (Blinkit): Feasto's acquisition of Blinkit enables rapid grocery and essentials delivery, expanding the platform's scope to quick commerce with up to 25,000 SKUs in select locations.

Restaurant Support: Feasto provides restaurants with advertising opportunities, premium listings, and consultancy services to enhance visibility and operations. Hyperpure supplies ingredients, and Feasto Gold offers loyalty programs to boost customer retention.

This flow is supported by Feasto's technology stack, which ensures scalability, reliability, and a user-friendly experience. The company's focus on operational excellence and data-driven strategies optimizes each step, contributing to its competitive edge.

User Types of Feasto

Feasto serves multiple user types, each playing a distinct role in its ecosystem. These are:

Customers

Description: End-users who use Feasto to discover restaurants, order food, or reserve tables. They range from urban millennials to families seeking convenient dining solutions.

Role: Customers drive demand by placing orders, providing reviews, and participating in loyalty programs like Feasto Gold. They value accessibility, affordability, and quality, as emphasized by Feasto's QAAA model (Quality, Accessibility, Affordability, and Portfolio).

Demographics: Primarily urban users in metro, Tier I, and Tier II cities across 25 countries, including India, the USA, Australia, and others. The platform's intuitive design and localization cater to diverse age groups and preferences.

Restaurants

Description: Partner restaurants, cafes, and eateries listed on Feasto's platform, ranging from small local vendors to large chains.

Role: Restaurants fulfill orders, pay commissions to Feasto, and benefit from increased visibility through listings, advertisements, and user reviews.Feasto's Hyperpure service supplies ingredients, and consultancy services help optimize operations.

Scale: Over 276,000 restaurant partners, leveraging Feasto's extensive network for customer reach.

Delivery Partners

Description: Independent contractors who handle last-mile delivery, ensuring orders reach customers quickly and reliably.

Role: Delivery partners are critical to Feasto's logistics, with algorithms assigning orders based on proximity and availability. They are not employees but partners, allowing flexibility in accepting or declining jobs. Their efficiency contributes to Feasto's 12.5-minute average delivery time.

Support: Feasto provides delivery partners with tools like Feasto Trace (from the Sparse Labs acquisition) for tracking and logistics optimization.

Business Clients (Hyperpure and Advertising)

Description: Restaurants and businesses that use Feasto's B2B services, such as Hyperpure for ingredient supplies or advertising for premium listings.

Role: These clients contribute to Feasto's revenue through procurement services and paid promotions. Hyperpure ensures high-quality ingredients, while advertising enhances restaurant visibility.

Feasto Gold Members

Description: Subscribed users who pay for premium services, gaining benefits like discounts, free deliveries, or exclusive offers.

Role: These users drive recurring revenue through subscription fees and increase platform engagement through loyalty programs. Introduced in 2018, Feasto Gold enhances customer retention.

Quick Commerce Users (Blinkit)

Description: Customers using Feasto's quick commerce arm, Blinkit, for grocery and essentials delivery.

Role: These users expand Feasto's market beyond food delivery, leveraging the same platform for rapid delivery of non-food items. Blinkit's integration adds diversity to Feasto's user base.

Feasto's internal flow is a well-orchestrated process involving customer discovery, order placement, restaurant preparation, delivery partner logistics, and feedback, all powered by advanced front-end and back-end technologies. Its user types include customers, restaurants, delivery partners, business clients, Feasto Gold members, and quick commerce users, each contributing to Feasto's ecosystem of reliability, accessibility, and growth. This multi-faceted approach, supported by data-driven strategies and acquisitions like Blinkit, has solidified Feasto's leadership in the food-tech and quick commerce industries.

Key Entities for Feasto-like Spring Boot Application

To implement a Spring Boot REST API application similar to Feasto's food delivery platform, you need to design entities that represent the core components of the system. Below, I outline the key entities based on Feasto's internal flow and user types, focusing on a simplified model for restaurant discovery, food ordering, and delivery. These entities will form the foundation of your database schema and REST API structure.

These entities cover the core functionalities of restaurant discovery, order management, delivery, and user interactions. Each entity is described with its primary attributes to capture the essential relationships and data.

Core User and Restaurant Entities

User Entity

Represents customers who browse restaurants, place orders, or leave reviews.

Attributes:

- userId (Primary Key, Long)
- name (String)
- email (String, unique)
- phoneNumber (String)
- password (String, encrypted)
- address (String or embedded Address entity)
- role (Enum: CUSTOMER, ADMIN, etc.)
- createdAt (Timestamp)
- updatedAt (Timestamp)

Relationships:

- One-to-Many with Order (a user can place multiple orders)
- One-to-Many with Review (a user can write multiple reviews)

Restaurant Entity

Represents partner restaurants listed on the platform.

Attributes:

- restaurantId (Primary Key, Long)
- name (String)
- description (String)
- address (String or embedded Address entity)
- phoneNumber (String)
- cuisineType (String or List<String>)
- rating (Double, calculated from reviews)
- isActive (Boolean)
- createdAt (Timestamp)
- updatedAt (Timestamp)

Relationships:

- One-to-Many with MenuItem (a restaurant has multiple menu items)
- One-to-Many with Order (a restaurant receives multiple orders)
- One-to-Many with Review (a restaurant can have multiple reviews)

Order Management Entities

MenuItem Entity

Represents food items offered by a restaurant.

Attributes:

- menuItemId (Primary Key, Long)
- restaurantId (Foreign Key, Long, links to Restaurant)
- name (String)
- description (String)
- price (Double)
- category (String, e.g., Appetizer, Main Course, Dessert)
- isAvailable (Boolean)
- imageUrl (String, optional for menu item image)

Relationships:

- Many-to-One with Restaurant (a menu item belongs to one restaurant)
- Many-to-Many with Order (via OrderItem, as orders can include multiple menu items)

Order Entity

Represents a customer's food order.

Attributes:

- orderId (Primary Key, Long)
- userId (Foreign Key, Long, links to User)
- restaurantId (Foreign Key, Long, links to Restaurant)
- deliveryPartnerId (Foreign Key, Long, links to DeliveryPartner, nullable)
- orderStatus (Enum: PLACED, PREPARING, OUT_FOR_DELIVERY, DELIVERED, CANCELLED)
- totalAmount (Double)
- deliveryAddress (String or embedded Address entity)
- orderTime (Timestamp)
- deliveryTime (Timestamp, nullable)

Relationships:

- Many-to-One with User (an order is placed by one user)
- Many-to-One with Restaurant (an order is fulfilled by one restaurant)
- Many-to-One with DeliveryPartner (an order is delivered by one delivery partner)
- One-to-Many with OrderItem (an order contains multiple items)

OrderItem Entity

Represents individual items within an order.

Attributes:

- orderItemId (Primary Key, Long)
- orderId (Foreign Key, Long, links to Order)
- menuItemId (Foreign Key, Long, links to MenuItem)
- quantity (Integer)
- price (Double, price at the time of order)

Relationships:

- Many-to-One with Order (an order item belongs to one order)
- Many-to-One with MenuItem (an order item corresponds to one menu item)

Delivery and Support Entities

DeliveryPartner Entity

Represents delivery personnel responsible for order delivery.

Attributes:

- deliveryPartnerId (Primary Key, Long)
- name (String)
- phoneNumber (String)
- email (String, unique)
- vehicleDetails (String, optional)
- isAvailable (Boolean)
- currentLocation (String or embedded Location entity, e.g., latitude/longitude)
- createdAt (Timestamp)
- updatedAt (Timestamp)

Relationships:

- One-to-Many with Order (a delivery partner can handle multiple orders)

Review Entity

Represents customer feedback for restaurants or orders.

Attributes:

- reviewId (Primary Key, Long)
- userId (Foreign Key, Long, links to User)
- restaurantId (Foreign Key, Long, links to Restaurant)
- orderId (Foreign Key, Long, links to Order, optional)
- rating (Integer, e.g., 1 to 5)
- comment (String)
- reviewTime (Timestamp)

Relationships:

- Many-to-One with User (a review is written by one user)
- Many-to-One with Restaurant (a review is for one restaurant)
- Many-to-One with Order (a review may be tied to a specific order, optional)

Address Entity

Represents delivery or restaurant addresses.

Attributes:

- addressId (Primary Key, Long, if standalone)
- street (String)
- city (String)
- state (String)
- postalCode (String)
- country (String)
- latitude (Double, for geolocation)
- longitude (Double, for geolocation)

Relationships:

- Many-to-One with User (a user can have multiple saved addresses)
- Many-to-One with Restaurant (a restaurant has one address)
- Many-to-One with Order (an order has one delivery address)

Payment and Loyalty Entities

Payment Entity

Represents payment transactions for orders.

Attributes:

- paymentId (Primary Key, Long)
- orderId (Foreign Key, Long, links to Order)
- userId (Foreign Key, Long, links to User)
- amount (Double)
- paymentMethod (Enum: ONLINE, COD, UPI, CARD)
- paymentStatus (Enum: PENDING, COMPLETED, FAILED)
- transactionId (String, unique)
- paymentTime (Timestamp)

Relationships:

- Many-to-One with Order (a payment is tied to one order)
- Many-to-One with User (a payment is made by one user)

LoyaltyProgram Entity

Represents premium membership or loyalty programs.

Attributes:

- loyaltyId (Primary Key, Long)
- userId (Foreign Key, Long, links to User)
- membershipType (Enum: GOLD, BASIC, etc.)
- startDate (Timestamp)
- endDate (Timestamp)
- benefits (String or List<String>, e.g., free delivery, discounts)

Relationships:

- Many-to-One with User (a user can have one active loyalty membership)

Notes on Implementation: Database: Use a relational database like MySQL to store these entities, with JPA/Hibernate in Spring Boot for ORM.t. Relationships: Define relationships using JPA annotations (@OneToMany, @ManyToOne, @ManyToMany) to ensure proper mapping. For example, use a join table for the Order-MenuItem relationship via OrderItem. Scalability: Feasto's platform handles high traffic, so design with scalability in mind. Use Spring Boot's caching (e.g., Redis) for frequently accessed data like restaurant listings or menus.

User Workflows and API Design

To implement a Feasto-like Spring Boot REST API application, understanding the workflow for each user type is crucial for designing the system's functionality. Below, I outline the workflows for each user type identified in Feasto Gold Members, and Quick Commerce Users). Each workflow describes the key actions and interactions with the platform, aligned with Feasto's operational flow. These workflows will help you define the API endpoints, business logic, and database interactions needed for your application.

Customer Workflow

Registration/Login → Restaurant Discovery → Order Placement → Order Tracking → Payment → Feedback

Restaurant Workflow

Onboarding → Menu Management → Order Receipt → Order Fulfillment → Feedback Review

Delivery Partner Workflow

Registration/Login → Availability → Order Assignment → Delivery Execution → Earnings Tracking

Business Client Workflow

Hyperpure (Ingredient Supply) → Advertising → Analytics

Feasto Gold Member Workflow

Subscription → Order Placement → Renewal/Expiration

Quick Commerce User Workflow

Product Discovery → Order Placement → Delivery → Feedback

Complete API Endpoint Reference

TCustomer Workflow APIs

These endpoints handle customer interactions like registration, restaurant discovery, order placement, tracking, payment, and feedback.

HTTP Method	Endpoint	Description	Request Body/Params	Response
POST	/api/users/register	Register a new customer	JSON: {name, email, phoneNumber, password, address}	User details with ID
POST	/api/users/login	Authenticate and login customer	JSON: {email, password}	JWT token
GET	/api/restaurants	Search and discover restaurants (with filters)	Query params: location, cuisine, rating	List of Restaurant objects
GET	/api/restaurants/{restaurantId}/menu	Get menu items for a specific restaurant	Path param: restaurantId	List of MenuItem objects
POST	/api/orders	Place a new order	JSON: {userId, restaurantId, items: [{menuItemId, quantity}], deliveryAddress, paymentMethod}	Order details with ID
GET	/api/orders/{orderId}	Track a specific order	Path param: orderId	Order details including status and tracking info
GET	/api/users/{userId}/orders	Get all orders for a customer	Path param: userId	List of Order objects
POST	/api/payments	Process payment for an order	JSON: {orderId, amount, paymentMethod}	Payment details with status
POST	/api/reviews	Submit a review for a restaurant/order	JSON: {userId, restaurantId, orderId (optional), rating, comment}	Review details
GET	/api/reviews/restaurant/{restaurantId}	Get reviews for a restaurant	Path param: restaurantId	List of Review objects

Restaurant Workflow APIs

These endpoints manage restaurant onboarding, menu updates, order handling, and feedback review.

HTTP Method	Endpoint	Description	Request Body/Params	Response
POST	/api/restaurants/register	Onboard a new restaurant	JSON: {name, description, address, phoneNumber, cuisineType}	Restaurant details with ID
POST	/api/restaurants/{restaurantId}/menu	Add a new menu item	Path param: restaurantId; JSON: {name, description, price, category}	MenuItem details
PUT	/api/restaurants/{restaurantId}/menu/{menuItemId}	Update a menu item	Path params: restaurantId, menuItemId; JSON: updated fields	Updated MenuItem
DELETE	/api/restaurants/{restaurantId}/menu/{menuItemId}	Delete a menu item	Path params: restaurantId, menuItemId	Success message
GET	/api/restaurants/{restaurantId}/orders	Get all orders for the restaurant	Path param: restaurantId	List of Order objects
PUT	/api/orders/{orderId}/status	Update order status (e.g., PREPARING, READY)	Path param: orderId; JSON: {status}	Updated Order
GET	/api/restaurants/{restaurantId}/reviews	Get reviews for the restaurant	Path param: restaurantId	List of Review objects
GET	/api/restaurants/{restaurantId}/analytics	Get analytics (order volume, ratings)	Path param: restaurantId	Analytics data (custom object)

Delivery Partner Workflow APIs

These endpoints handle delivery partner registration, availability, order assignment, and execution.

HTTP Method	Endpoint	Description	Request Body/Params	Response
POST	/api/delivery-partners/register	Register a new delivery partner	JSON: {name, phoneNumber, email, vehicleDetails}	DeliveryPartner details with ID
POST	/api/delivery-partners/login	Authenticate and login delivery partner	JSON: {email, password}	JWT token
PUT	/api/delivery-partners/{deliveryPartnerId}/availability	Toggle availability and update location	Path param: deliveryPartnerId; JSON: {isAvailable, currentLocation: {lat, lng}}	Updated DeliveryPartner
GET	/api/delivery-partners/{deliveryPartnerId}/available-orders	Get nearby available orders for assignment	Path param: deliveryPartnerId	List of Order objects
POST	/api/orders/{orderId}/accept	Accept an assigned order	Path param: orderId; JSON: {deliveryPartnerId}	Updated Order with assignment
PUT	/api/orders/{orderId}/pickup	Mark order as picked up (status: OUT_FOR_DELIVERY)	Path param: orderId	Updated Order
PUT	/api/orders/{orderId}/deliver	Mark order as delivered	Path param: orderId; JSON: {paymentStatus (if COD)}	Updated Order and Payment
GET	/api/delivery-partners/{deliveryPartnerId}/earnings	Get earnings history	Path param: deliveryPartnerId	Earnings data (custom object)

Business Client Workflow APIs (Hyperpure and Advertising)

These endpoints support B2B services like ingredient supply and advertising.

HTTP Method	Endpoint	Description	Request Body/Params	Response
POST	/api/business-clients/register	Register for B2B services (extends Restaurant)	JSON: {restaurantId, serviceType (e.g., HYPERPURE, ADVERTISING)}	BusinessClient details
POST	/api/supply-orders	Place an ingredient supply order (hyperpure)	JSON: {restaurantId, items: [{itemName, quantity}], deliveryAddress}	SupplyOrder details
GET	/api/supply-orders/{supplyOrderId}	Track a supply order	Path param: supplyOrderId	SupplyOrder details
POST	/api/advertisements	Create an advertising campaign	JSON: {restaurantId, adType, budget, duration, targetAudience}	Advertisement details
PUT	/api/advertisements/{adId}	Update an ad campaign	Path param: adId; JSON: updated fields	Updated Advertisement
GET	/api/advertisements/{adId}/performance	Get ad performance analytics	Path param: adId	Performance data (custom object)
GET	/api/business-clients/{clientId}/analytics	Get overall B2B analytics	Path param: clientId	Analytics data (custom object)

Feasto **Gold Member Workflow APIs**

These endpoints manage premium subscriptions and benefits application.

HTTP Method	Endpoint	Description	Request Body/Params	Response
POST	/api/loyalty/subscribe	Subscribe to Feasto Gold	JSON: {userId, membershipType, paymentDetails}	LoyaltyProgram details
GET	/api/loyalty/{loyaltyId}	Get membership details	Path param: loyaltyId	LoyaltyProgram details
PUT	/api/loyalty/{loyaltyId}/renew	Renew membership	Path param: loyaltyId; JSON: {paymentDetails}	Updated LoyaltyProgram
GET	/api/users/{userId}/loyalty	Check active membership for a user	Path param: userId	LoyaltyProgram details or null
POST	/api/orders/with-loyalty	Place order with loyalty benefits (extends /api/orders)	JSON: same as order placement + loyaltyId	Order details with applied discounts

Quick Commerce User Workflow APIs (Blinkit-like)

These endpoints handle quick commerce for groceries and essentials, similar to food orders but with products.

HTTP Method	Endpoint	Description	Request Body/Params	Response
GET	/api/products	Search and discover products (with filters)	Query params: category, location	List of Product objects
GET	/api/products/{productId}	Get details for a specific product	Path param: productId	Product details
POST	/api/quick-commerce/orders	Place a quick commerce order	JSON: {userId, items: [{productId, quantity}], deliveryAddress, paymentMethod}	Order details with ID
GET	/api/quick-commerce/orders/{orderId}	Track a quick commerce order	Path param: orderId	Order details including status
GET	/api/users/{userId}/quick-commerce/orders	Get all quick commerce orders for a user	Path param: userId	List of Order objects
POST	/api/quick-commerce/reviews	Submit a review for a quick commerce order	JSON: {userId, orderId, rating, comment}	Review details
POST	/api/quick-commerce/payments	Process payment for quick commerce order	JSON: {orderId, amount, paymentMethod}	Payment details

- **Implementation Notes:** Error Handling: Implement global exception handling for validation errors, not found, etc., returning standard HTTP status codes (e.g., 400 Bad Request, 404 Not Found). Pagination and Filtering: For list endpoints (e.g., GET /api/restaurants), add query params for page, size, and sort using Spring Data's Pageable. Real-Time Features: For tracking, integrate WebSockets (e.g., /ws/orders/{orderId}) to push status updates. Validation: Use @Valid and Bean Validation for request bodies. Database Integration: Map to entities via services and repositories (e.g., OrderService, OrderRepository). Additional Endpoints: Admin endpoints like /api/admin/users for management can be added if needed.