Interviews: Go

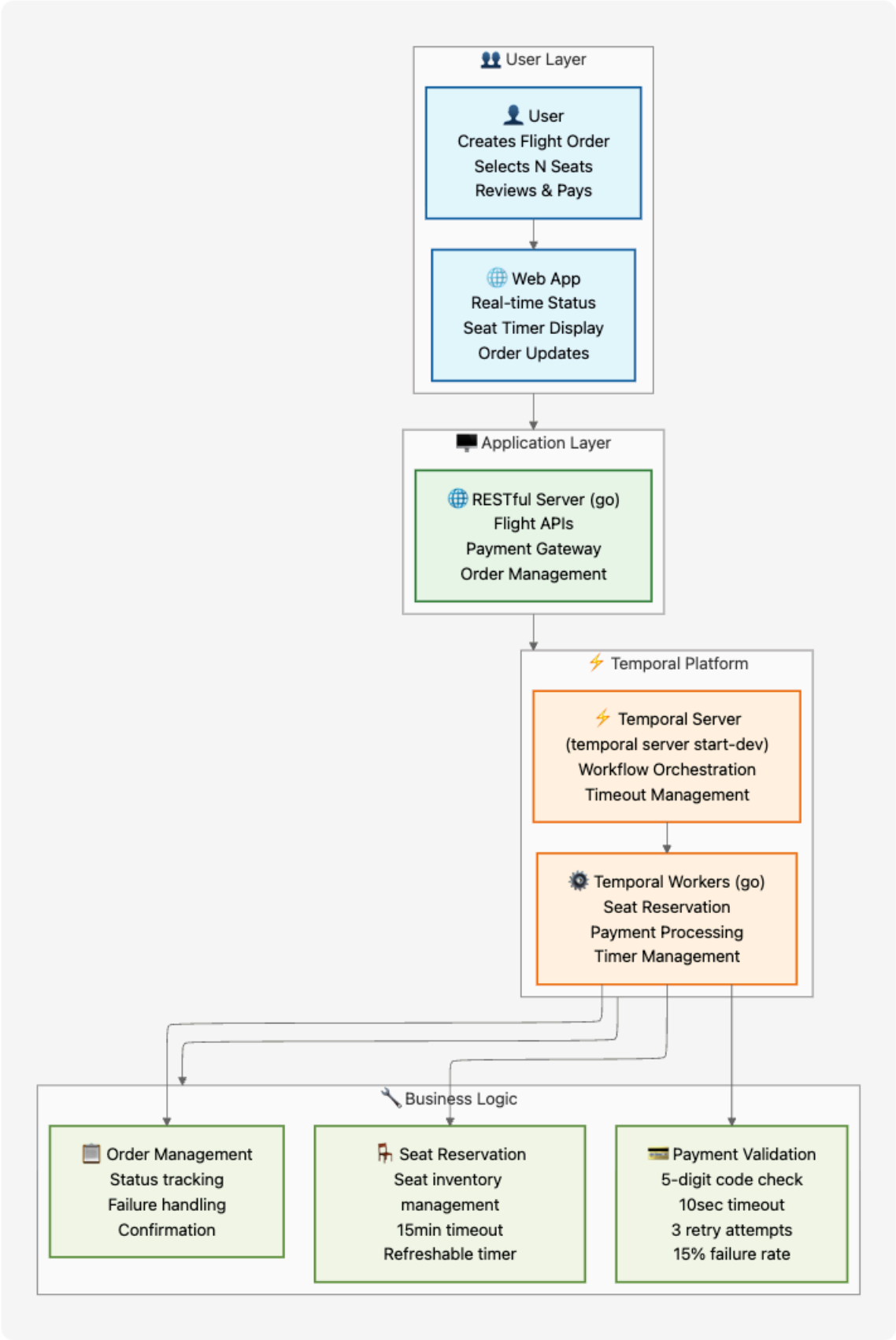
Owner	Gil Ron
Tags	

Flight Booking System - Temporal Architecture

This diagram shows the flight booking flow from User through Web App, RESTful Server, Temporal, and finally to Workers.

User Flow Exercise

Scenario: Flight seat reservation and payment system with timeouts and validation.


User Layer
User
Creates Flight Order
Selects N Seats
Reviews & Pays
Web App
Real-time Status
Seat Timer Display
Order Updates
Application Layer
RESTful Server (go)
Flight APIs
Payment Gateway
Order Management
Temporal Platform
Temporal Server
(temporal server start-dev)
Workflow Orchestration
Timeout Management
Temporal Workers (go)
Seat Reservation
Payment Processing
Timer Management
Business Logic
Order Management
Status tracking
Failure handling
Confirmation
Seat Reservation
Seat inventory
management
15min timeout
Refreshable timer
Payment Validation
5-digit code check
10sec timeout
3 retry attempts
15% failure rate

Architecture Components

- **User**: Customer booking flights, selecting seats, and making payments
- **Web App**: Frontend showing real-time seat timer, order status, and booking interface
- **RESTful Server (go)**: Go-based APIs for flight management, payment gateway integration, and order processing
- **Temporal Server**: Orchestrates complex booking workflows with timeouts and state management
- **Workers (go)**: Execute specific booking activities and handle business logic

Business Logic

- 🪑 **Seat Reservation**: Manages seat inventory and 15-minute seat holds with refreshable timers when user updates selection
- 💳 **Payment Validation**: Processes 5-digit payment codes with 10-second timeout, 3 retry attempts, 15% failure rate simulation
- 📋 **Order Management**: Tracks order status, handles failures, sends confirmations

Mandatory Temporal Workflows

- At the very least, you should have a temporal workflow that implements the entire order

Detailed User Flow

1. **Create Flight Order**: User initiates booking process
2. **Select N Seats**: User chooses seats → **Seat Reservation Workflow starts** (15min timer)
3. **Review Order**: User can see timer countdown and modify seats (timer refreshes on changes)
4. **Pay with 5-digit Code**: User enters payment code → **Payment Code Validation**

5. **Code Validation**: System checks code within 10 seconds
    - **Success (85%)**: Charge user → **Order Management** → Confirmation message
    - **Failure (15%)**: Retry up to 3 times → After 3 failures, order fails with indicative message
6. **Real-time Updates**: User always sees seat timer and order status

Business Rules Implemented by Temporal

- ✅ **Seat inventory management** (implementation approach left to developer)
- ✅ 15-minute seat reservation with auto-release
- ✅ Timer refresh on seat updates
- ✅ 10-second payment validation timeout
- ✅ 3 retry attempts for failed payments
- ✅ 15% payment failure simulation
- ✅ Real-time status tracking
- ✅ Graceful failure handling with user feedback

Implementation Notes

Seat Management: The system must handle seat inventory internally - this is not delegated to external microservices. Implementation approach is flexible and could include:

- Custom data structures with conflict resolution
- In-memory state with persistence
- Temporal Entity Workflows for seat state management
- Traditional database with transaction management

Choice of approach is left to the implementer

The solid arrows show the primary request flow, while the dotted arrows show the response/result flow back to the user.